Simplify stream processing

with Python, Quix, and InfluxDB

Hello, nice to meet you! 👋

Tomas Neubauer

CTO & Co-founder, Quix

Previously McLaren technical lead

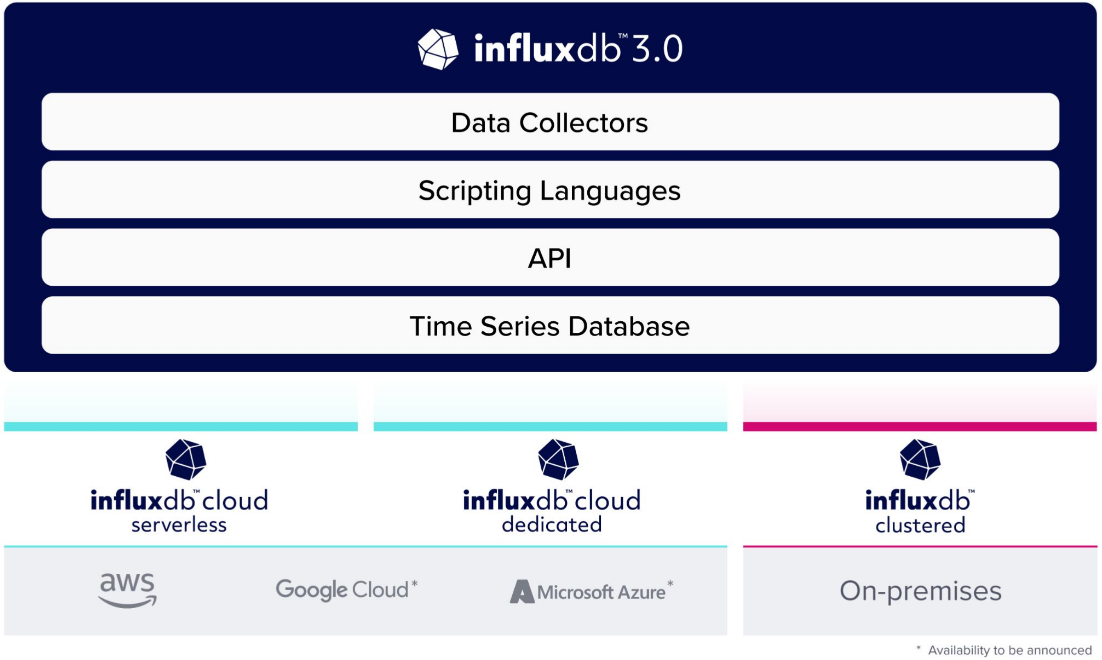
Racing background

Roots in real-time data processing in the most extreme, time-critical environment.

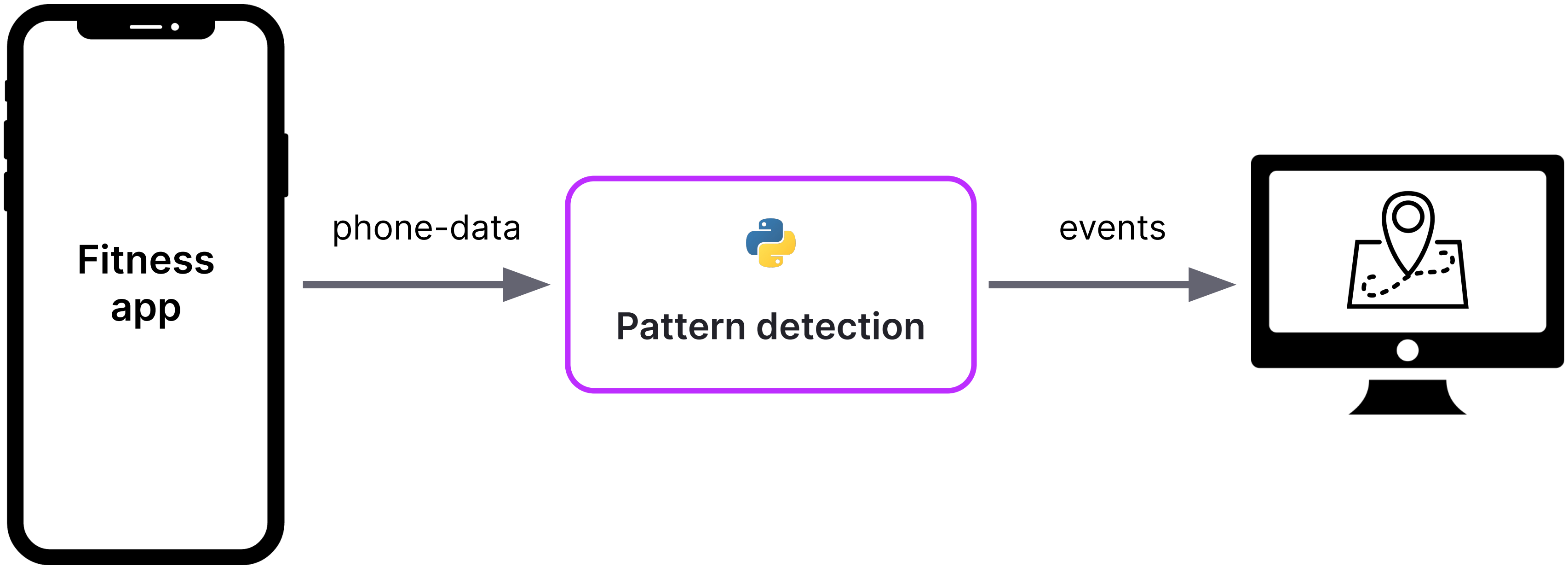
- 50,000 channels per car
- 1.5 kHz per channel
- 1,000s realtime models and simulations

What is InfluxDB?

InfluxDB. It's About Time.
Manage all types of time series data in a single, purpose-built database. Run at any scale in any environment in the cloud, on-premises, or at the edge.



Live demo!



Now raise your hand if you are using...

Kafka

Now raise your hand if you are using...

Streaming

Now raise your hand if you are using...

Python

Now raise your hand if you are using...

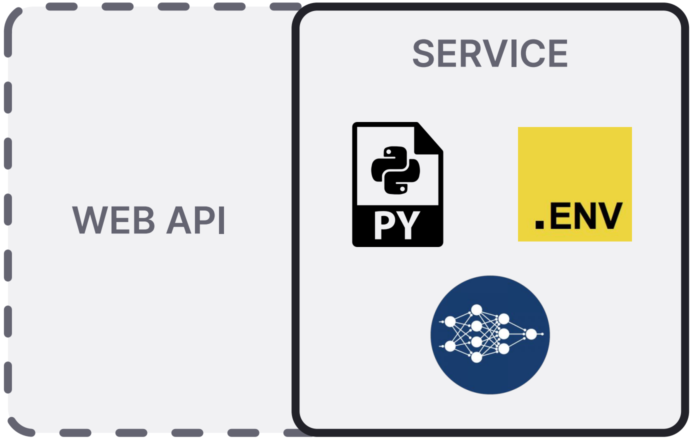
ML Deployment

REST API vs Streaming

ML Deployment with API

API REQUEST

gX	gY	gZ	gTotal
0.5	0.3	0.1	0.9



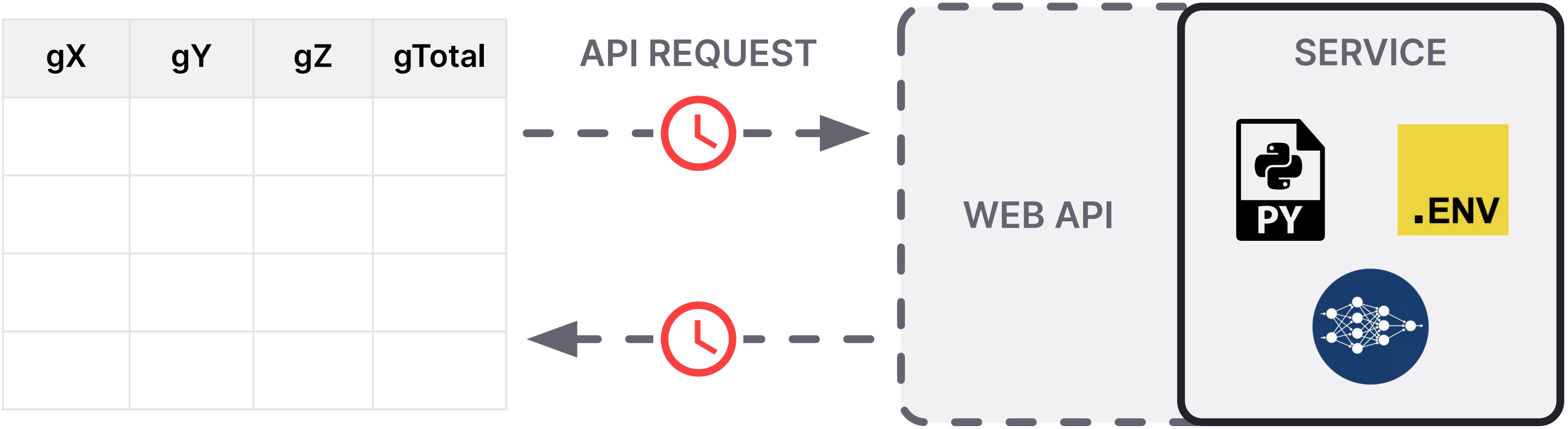
API RESPONSE

gX	gY	gZ	gTotal	Crash
0.5	0.3	0.1	0.9	1

Issues with REST APIs

REST API vs Streaming

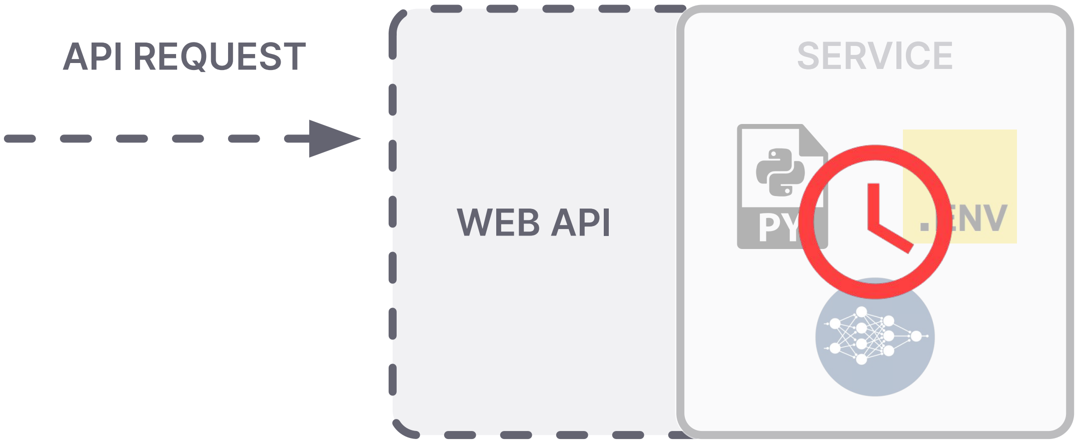
Problems with REST API



- CPU overhead
- Introducing delay
- Requests gets lost in case of service downtime or slow performance

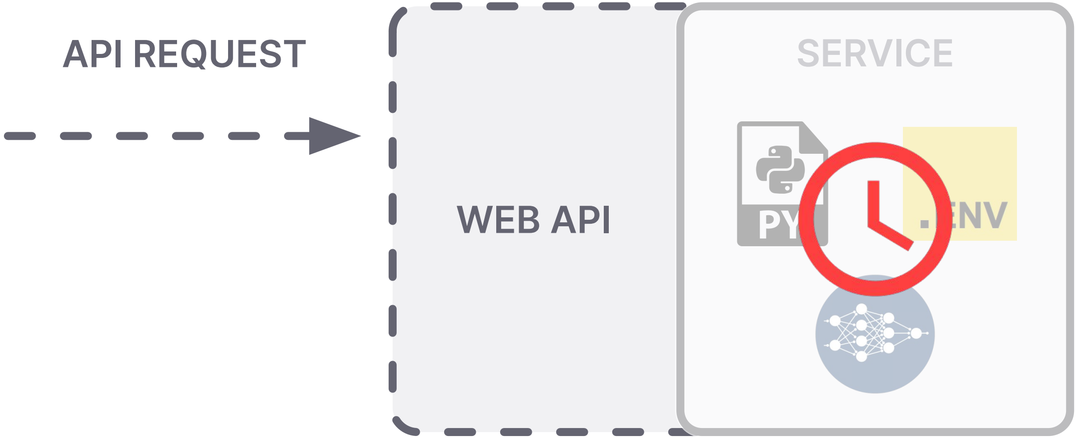
Problems with REST API

gX	gY	gZ	gTotal



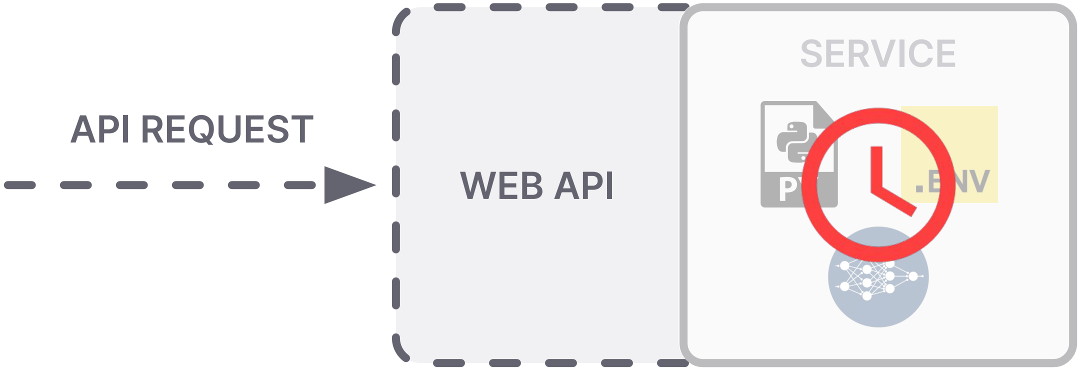
Problems with REST API

gX	gY	gZ	gTotal

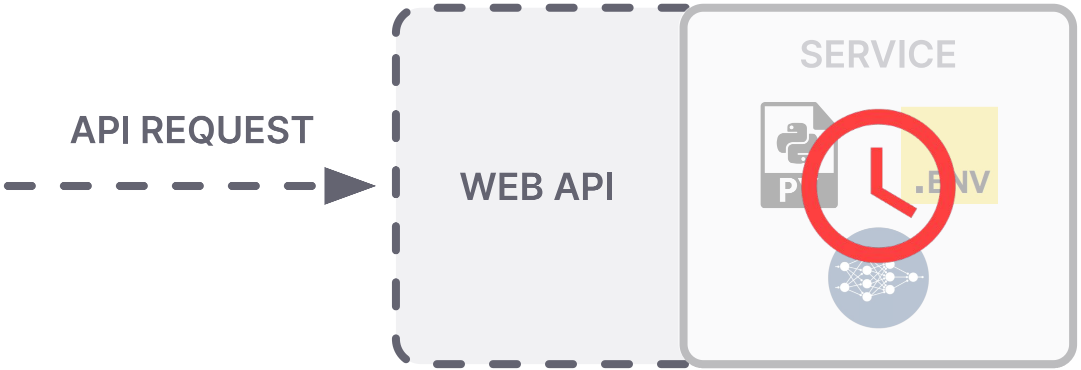


Problems with REST API

gX	gY	gZ	gTotal



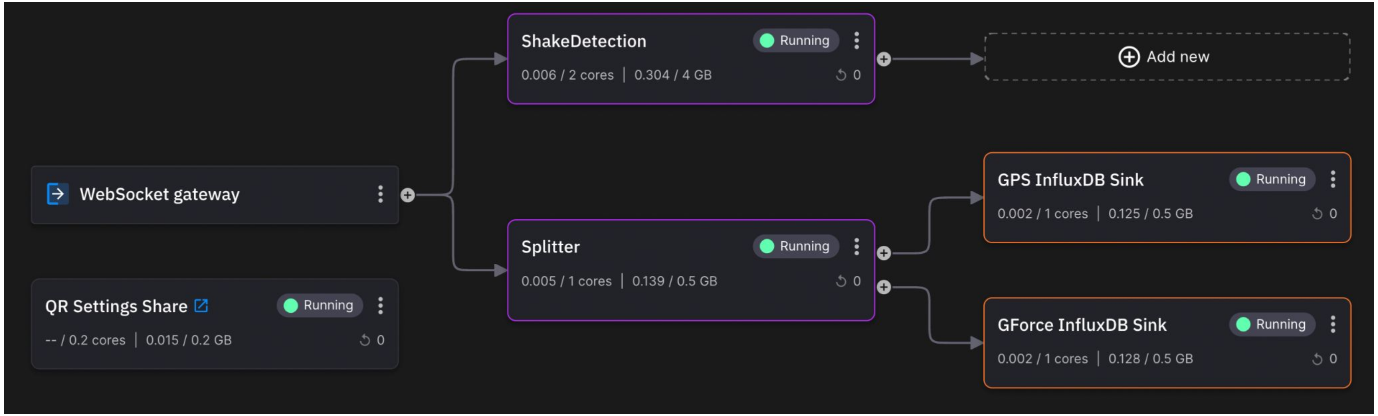
gX	gY	gZ	gTotal



Event streaming applications

What is an event streaming application?

- Built with Kafka & microservices
- Processes and transports data as continuous streams of events
  - Sensor data
  - Mouse clicks
  - Financial data
- Contains a pipelines to ingest → process → sink data



How to build event streaming apps

Event streaming architecture

When you build **event streaming applications** with **Kafka**, there are two options:

1. Just build an application with microservices uses the Kafka **producer** and **consumer** APIs directly
   a. combine Kubernetes with Kafka

2. Adopt a full-fledged **stream processing framework** (Flink, Spark streaming, Beam etc.)
   a. combine microservices in Kubernetes, Flink cluster, Flink jobs and Kafka

Kafka producer and consumer APIs

- Works for simple stuff like one-message-at-a-time processing
- No external dependencies like JVM

- Gets very **complicated** when **stateful processing** is needed like calculation aggregations or joining multiple streams

- **CI/CD overhead**
    - **Manage** your own **Kubernetes**
    - Own **build** and **release** pipelines

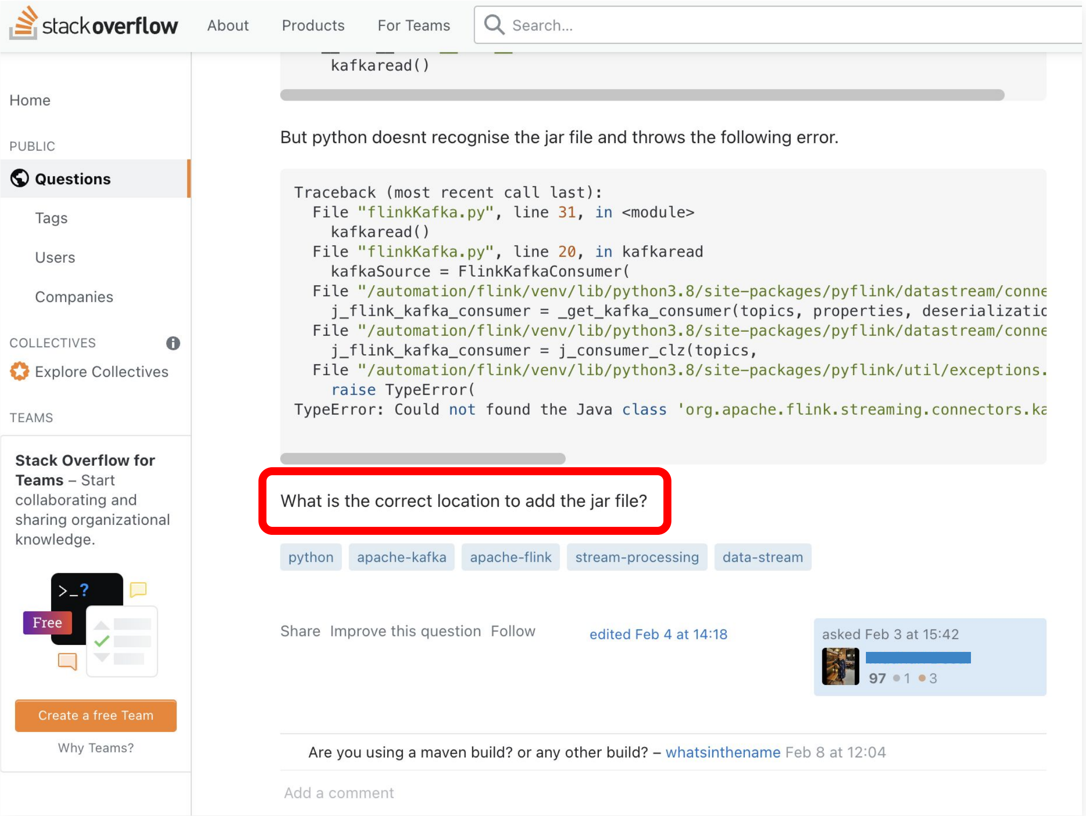
- Build own **monitoring** and **observability**

Stream processing frameworks

- Fully fledged stream processing frameworks solves stateful, more complex operations
- You get **CI/CD** and **observability** out of box for your **data pipelines**
  - but not **application microservices**

- Increased complexity in many dimensions:
  - Java dependency
  - Deployment gets difficult because code is not running on its own but in server side cluster (Flink cluster or Spark cluster)
  - Debugging is difficult
  - Performance optimization is difficult

JAR files...



stackoverflow
About
Products
For Teams
Search...
Home
PUBLIC
Questions
Tags
Users
Companies
COLLECTIVES
Explore Collectives
TEAMS
Stack Overflow for Teams – Start collaborating and sharing organizational knowledge.
>_?
Free
Create a free Team
Why Teams?
kafkaread()
But python doesnt recognise the jar file and throws the following error.
Traceback (most recent call last):
File "flinkKafka.py", line 31, in <module>
kafkaread()
File "flinkKafka.py", line 20, in kafkaread
kafkaSource = FlinkKafkaConsumer(
File "/automation/flink/venv/lib/python3.8/site-packages/pyflink/datastream/conne
j_flink_kafka_consumer = _get_kafka_consumer(topics, properties, deserializatic
File "/automation/flink/venv/lib/python3.8/site-packages/pyflink/datastream/conne
j_flink_kafka_consumer = j_consumer_clz(topics,
File "/automation/flink/venv/lib/python3.8/site-packages/pyflink/util/exceptions.
raise TypeError(
TypeError: Could not found the Java class 'org.apache.flink.streaming.connectors.ka
What is the correct location to add the jar file?
python
apache-kafka
apache-flink
stream-processing
data-stream
Share Improve this question Follow
edited Feb 4 at 14:18
asked Feb 3 at 15:42
97 1 3
Are you using a maven build? or any other build? – whatsinthename Feb 8 at 12:04
Add a comment

Connecting Flink to Kafka is difficult

```
CREATE TABLE country_target (
 country VARCHAR,
 avg_age BIGINT,
 nr_people BIGINT,
 PRIMARY KEY (country) NOT ENFORCED
) WITH (
  'connector' = 'upsert-kafka',
  'property-version' = 'universal',
  'properties.bootstrap.servers' = '<host>:<port>',
  'topic' = 'country_agg',
  'value.format' = 'json',
  'key.format' = 'json',
  'properties.security.protocol' = 'SSL',
  'properties.ssl.endpoint.identification.algorithm' = '',
  'properties.ssl.truststore.location' = '/settings/certs/client.truststore.jks',
  'properties.ssl.truststore.password' = 'password123',
  'properties.ssl.keystore.type' = 'PKCS12',
  'properties.ssl.keystore.location' = '/settings/certs/client.keystore.p12',
  'properties.ssl.keystore.password' = 'password123',
  'properties.ssl.key.password' = 'password123',
  'properties.group.id' = 'my-working-group'
);
```

SQL looks easy to use but...

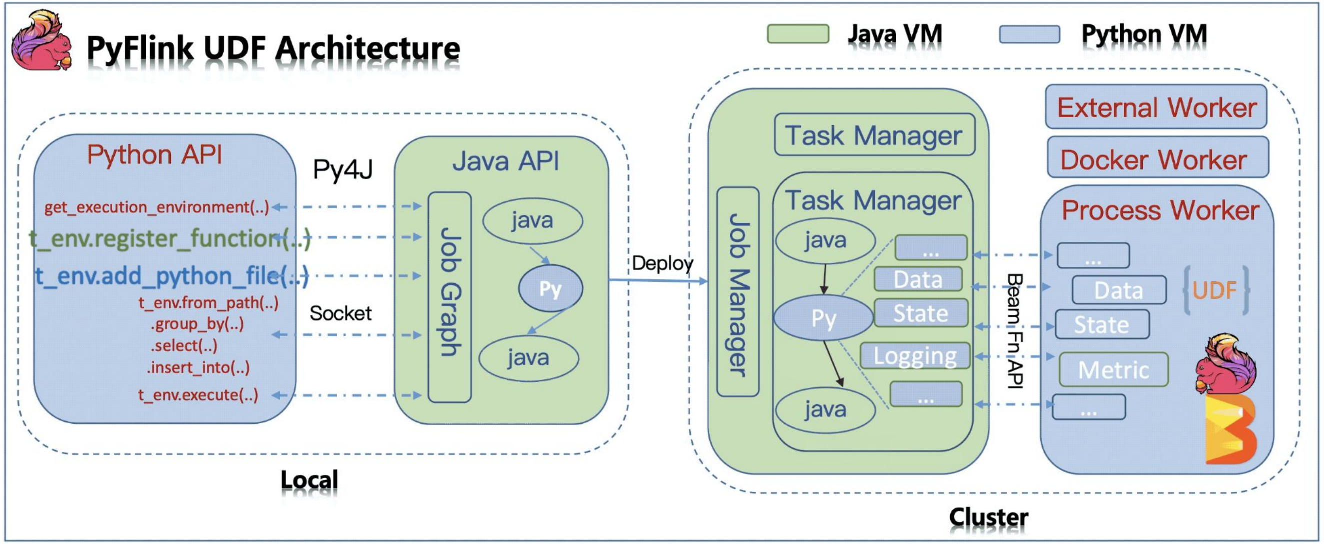
```
select * from country_target;
```

We should see something like this:

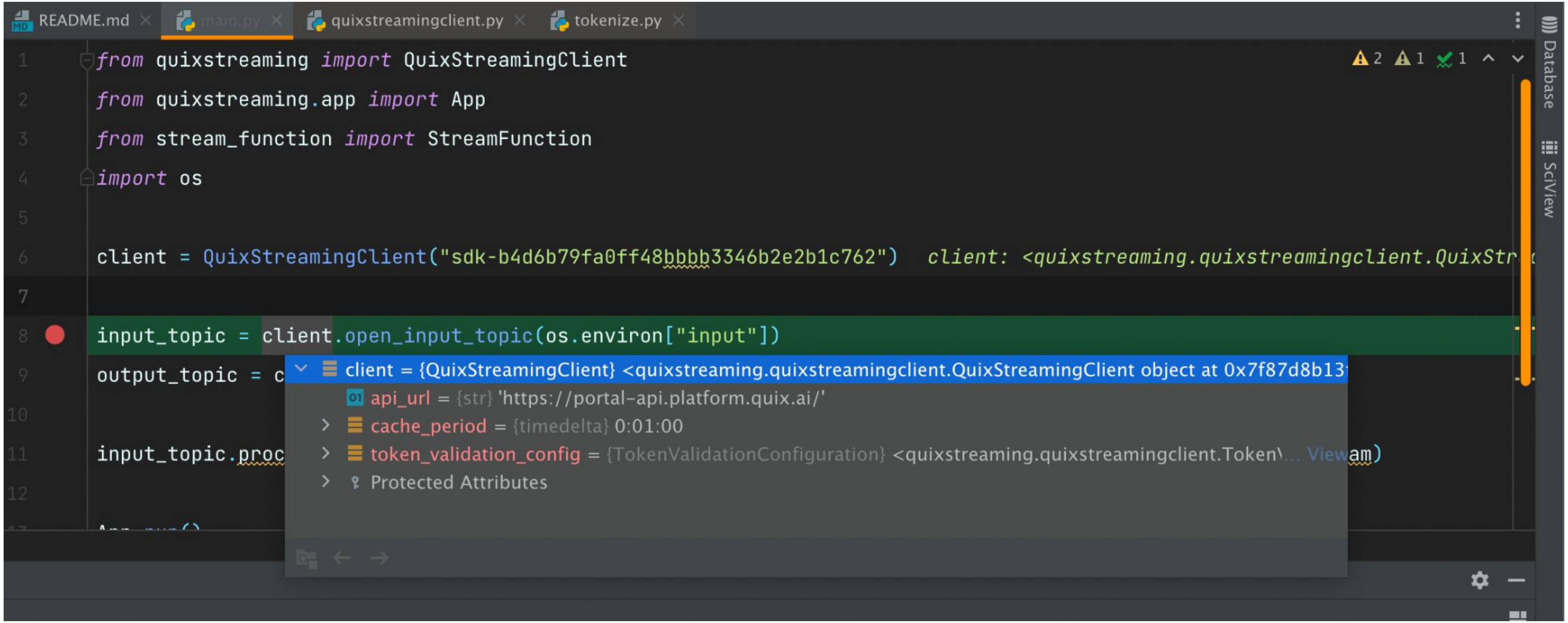
```
+/-          country        avg_age        nr_people
  +              USA             40                1
  +          England             35                1
  +            Italy             25                1
  -            Italy             25                1
  +            Italy             35                2
```

UDFs are nasty

- Poor development experience
  - Logs only accessible from server, no debugging possible
- Performance hit caused by interface between JVM and Python



DEBUGGING!!! 🐛🐛🐛



README.md
main.py
quixstreamingclient.py
tokenize.py
from quixstreaming import QuixStreamingClient
from quixstreaming.app import App
from stream_function import StreamFunction
import os
client = QuixStreamingClient("sdk-b4d6b79fa0ff48bbbb3346b2e2b1c762")
client: <quixstreaming.quixstreamingclient.QuixStr
input_topic = client.open_input_topic(os.environ["input"])
output_topic = c
input_topic.proc
client = {QuixStreamingClient} <quixstreaming.quixstreamingclient.QuixStreamingClient object at 0x7f87d8b13
api_url = {str} 'https://portal-api.platform.quix.ai/'
cache_period = {timedelta} 0:01:00
token_validation_config = {TokenValidationConfiguration} <quixstreaming.quixstreamingclient.Token\... View
Protected Attributes
Database
SciView

Building your own architecture is costly

8 months → 3 Months → 3 Weeks → 7 Days →

Build infrastructure

Technical complexities

Design complexities

Develop

Effort to build the first app

Data consistency and synchronisation

Release

Complex testing

Orchestration and management

Observe

Effectively monitoring and debugging

Platform team: 11 FTE

Engineering: 2 FTE + Data team: 2 FTE

Lower — **Cost & Risk** — Higher

One tool to build event streaming apps

Accelerated application development

Weeks → Hours → Minutes

Develop

Use free open-source connectors & code samples to develop faster. Use Python to process streams and get ML predictions.

Release

IaC: code, test and deploy event streaming applications with a single source of truth powered by Kafka, Docker and Git.

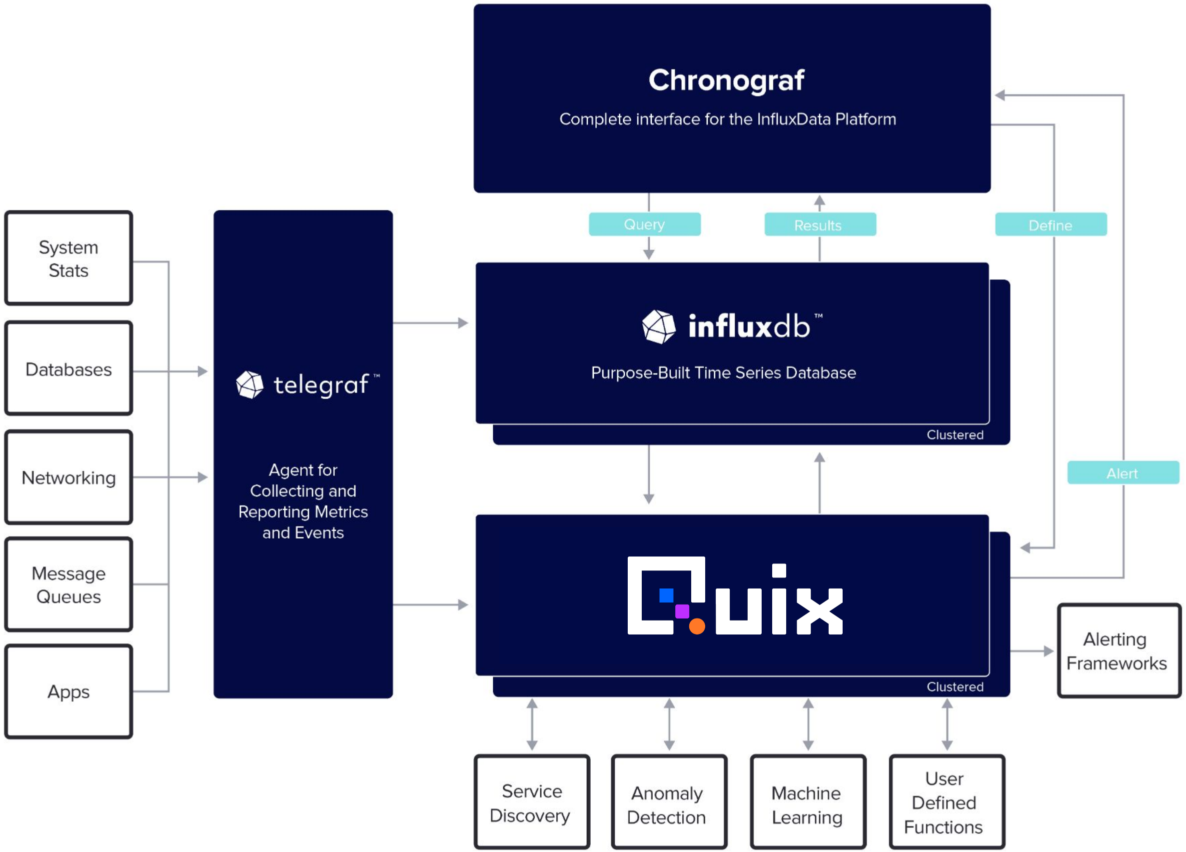
Observe

A suite of observability tools designed to give you in-depth insights into your event-driven architecture.

Engineering: 2 FTE + Data team: 2 FTE

Predictable Cost & Risk

Easily integrate with InfluxDB Cloud?



A new way to process streaming data

- Application development environment managing CI/CD and releases for **data pipelines, Kafka** and **microservices** in one tool

- Combining Kafka API approach with a **Python stream processing library**

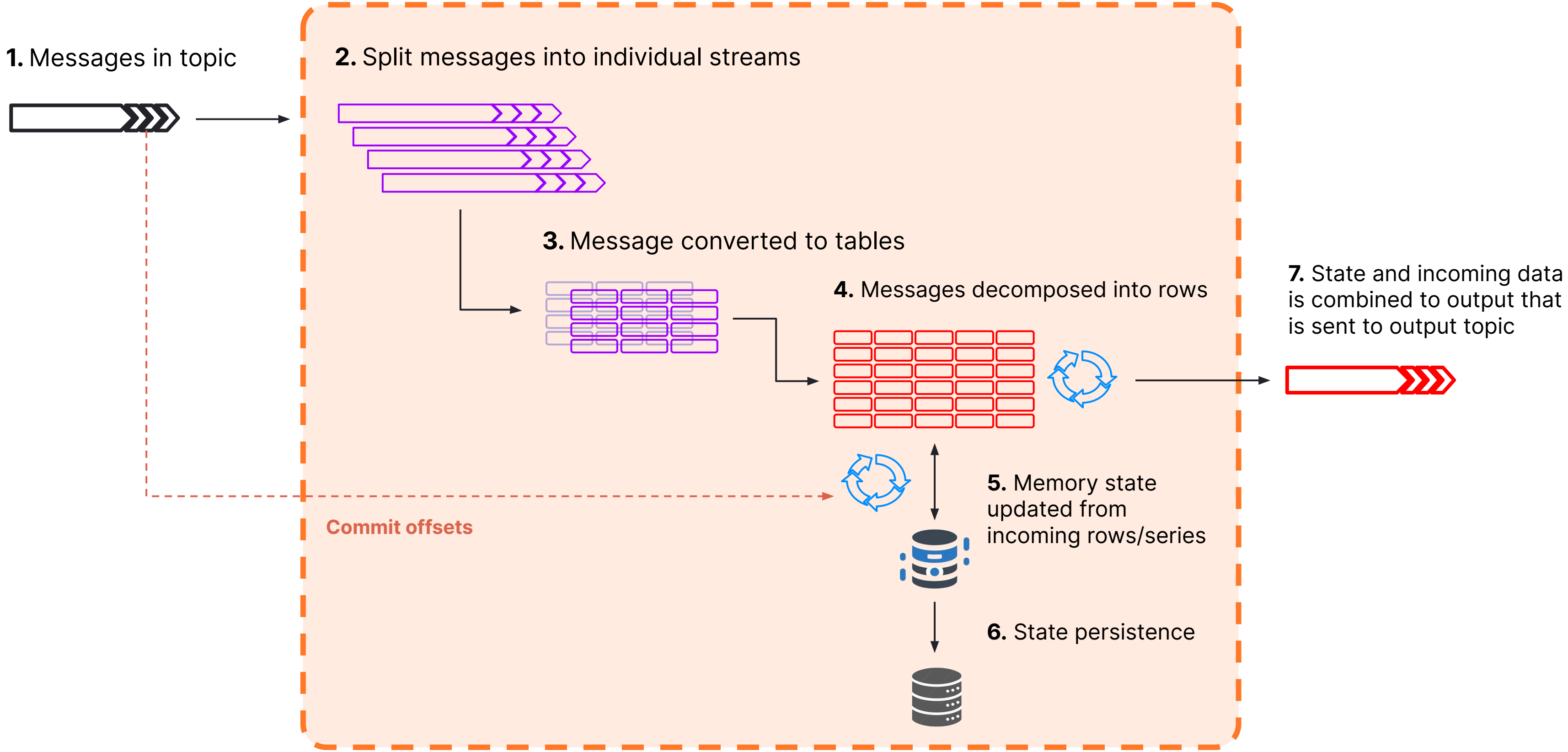
- **Standalone library** that runs:
  - Locally for development and debugging
  - In docker or in Kubernetes for production deployments at scale

- **Seamless integration** with external systems like **InfluxDB Cloud**

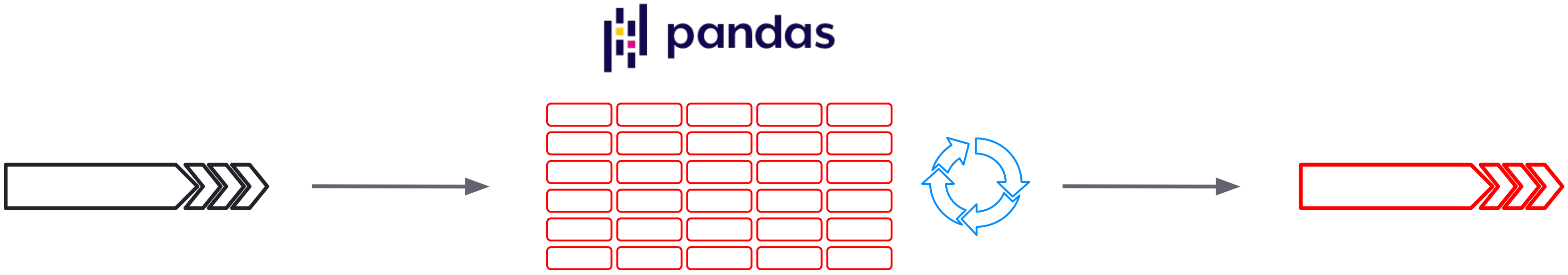
Quix Streams

Python Streaming DataFrames

Stateful processing with Pub & Sub client libraries



Quix Streams PySDF



1. Messages in topic

2. Messages decomposed as rows available via pandas API

3. Messages processed through pipeline defined as pandas operations. Output streamed to output topic.

- Automatic state management
- Automatic checkpointing
- Automatic message serialization/deserialization

Quix Streams PySDF API

```
# Define topics with serialization settings
input_topic = Topic("input_topic", value_deserializer=JSONDeserializer())
output_topic = Topic("output_topic", value_serializer=JSONSerializer())

# Define a StreamingDataframe to transform the data
sdf = StreamingDataFrame(topics=[input_topic])

# Select only "field_A", "field_B", "field_C" from the incoming message
sdf = sdf[['field_A', 'field_B', 'field_C']]

# Filter only messages with "field_A" > 5 and "field_B" < 4
sdf = sdf[(sdf['field_A'] > 5) & (sdf['field_B'] < 4)]

# Produce the result to the output topic
sdf = sdf.to_topic(output_topic)


# Run the dataframe
with Runner(broker_address="localhost:9092", consumer_group="test", auto_offset_r
    runner.run(sdf)
```

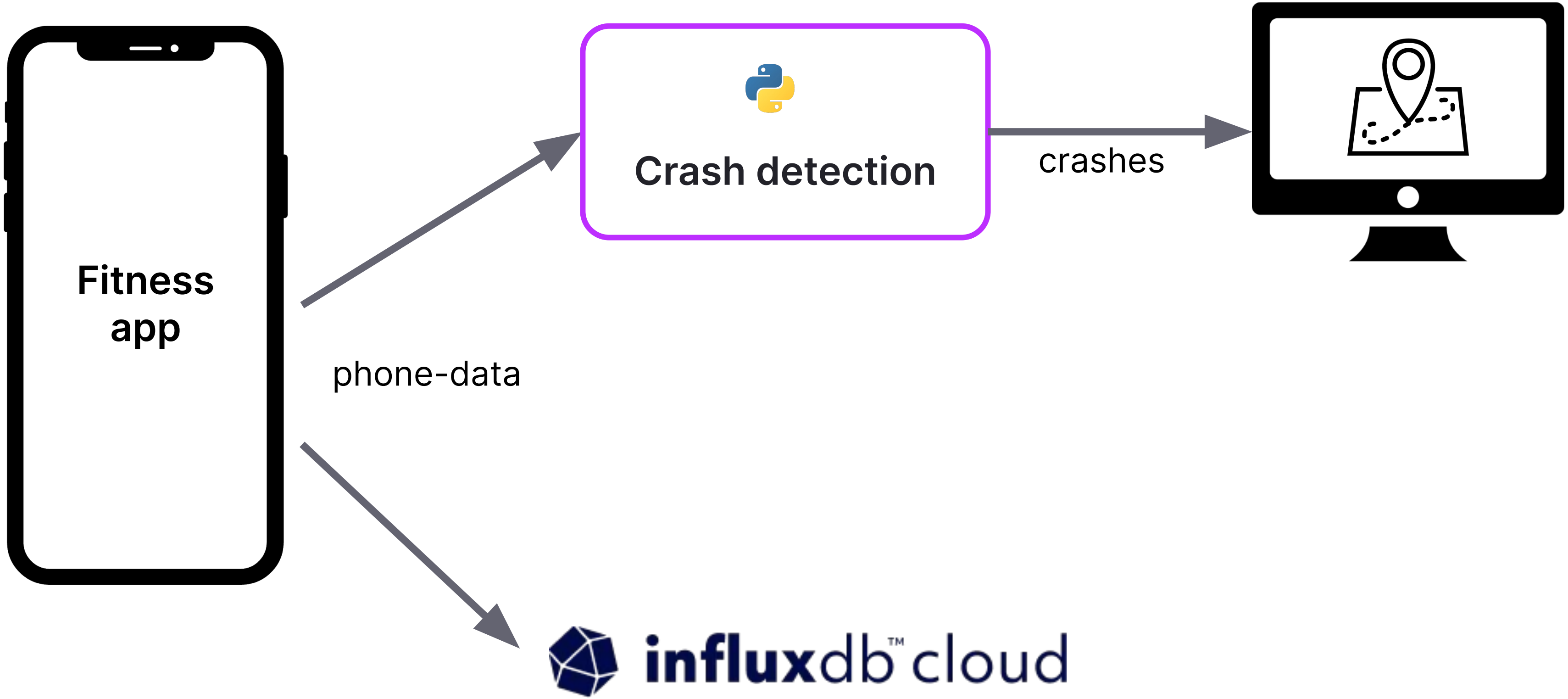
Quix Streams PySDF V1.0 features

```python
# Define topics with JSON deserialization
input_topic = Topic("input_topic", value_deserializer=JSONDeserializer())

# Define a StreamingDataframe
sdf = StreamingDataFrame(topics=[input_topic])

# Calculate sum of values in "A" over the tumbling window of size 10s
a_total_10s = sdf['A'].rolling(period=10, window_type='tumbling').sum()

# Update the current message with the result of window aggregation
sdf['A_total_10s'] = a_total_10s.value()

# Set additional window metadata
sdf['A_total_10s__start'] = a_total_10s.window_start()
sdf['A_total_10s__end'] = a_total_10s.window_end()
```

Demo

See it in action!

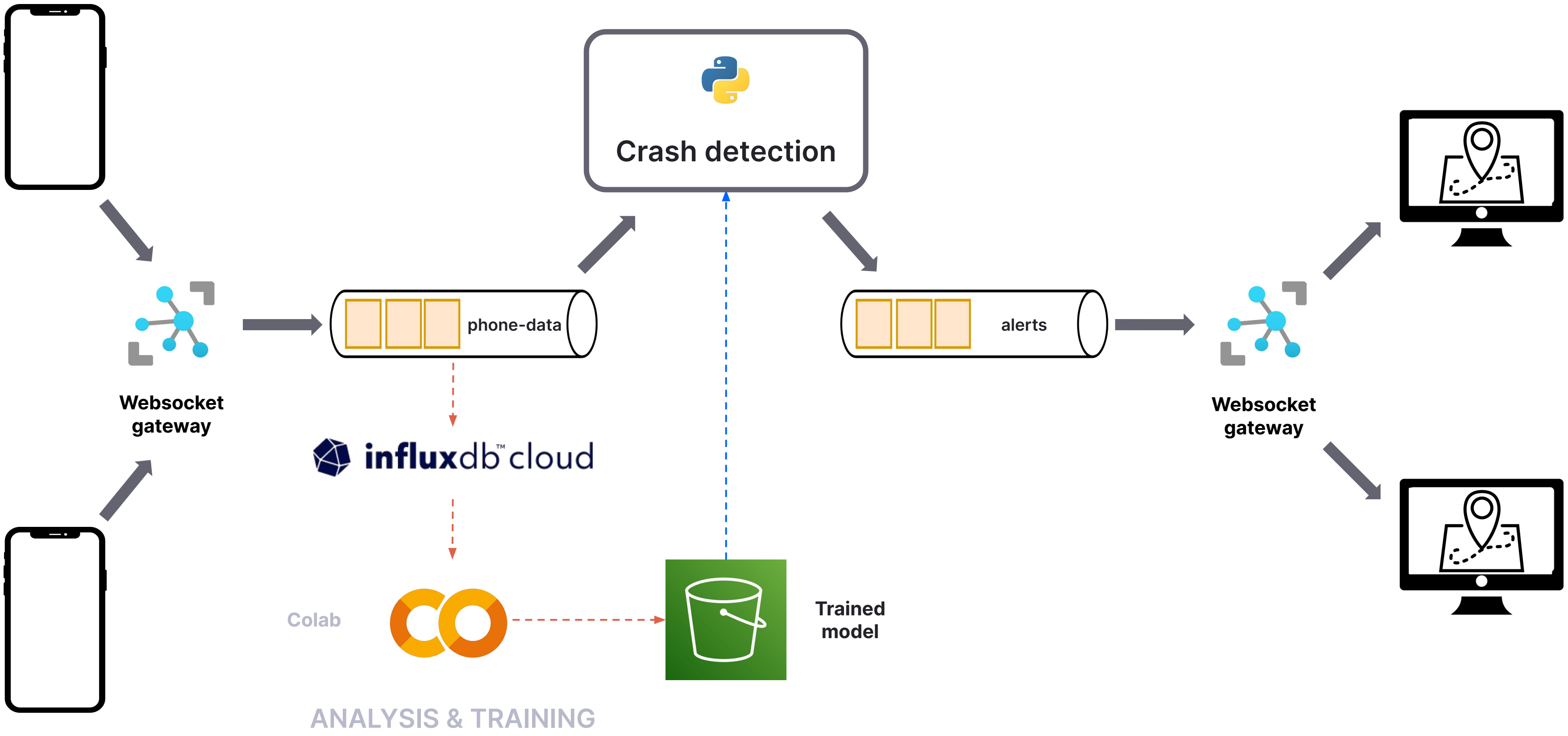
Use case 1: Real-time event detection app



Use case 2: Training real-time ML models with InfluxDB

Use case 2: Training real-time ML models with InfluxDB

Complete event streaming application architecture



How it works

Kafka + Kubernetes + Python

Our approach to stream processing

Containers

Containers running in Kubernetes scaling hand to hand with Kafka for compute scalability.

Kafka

Handle your data reliably and efficiently in memory with Kafka. Using Kafka partitions, replica system and persistence to deliver scalability and robustness.

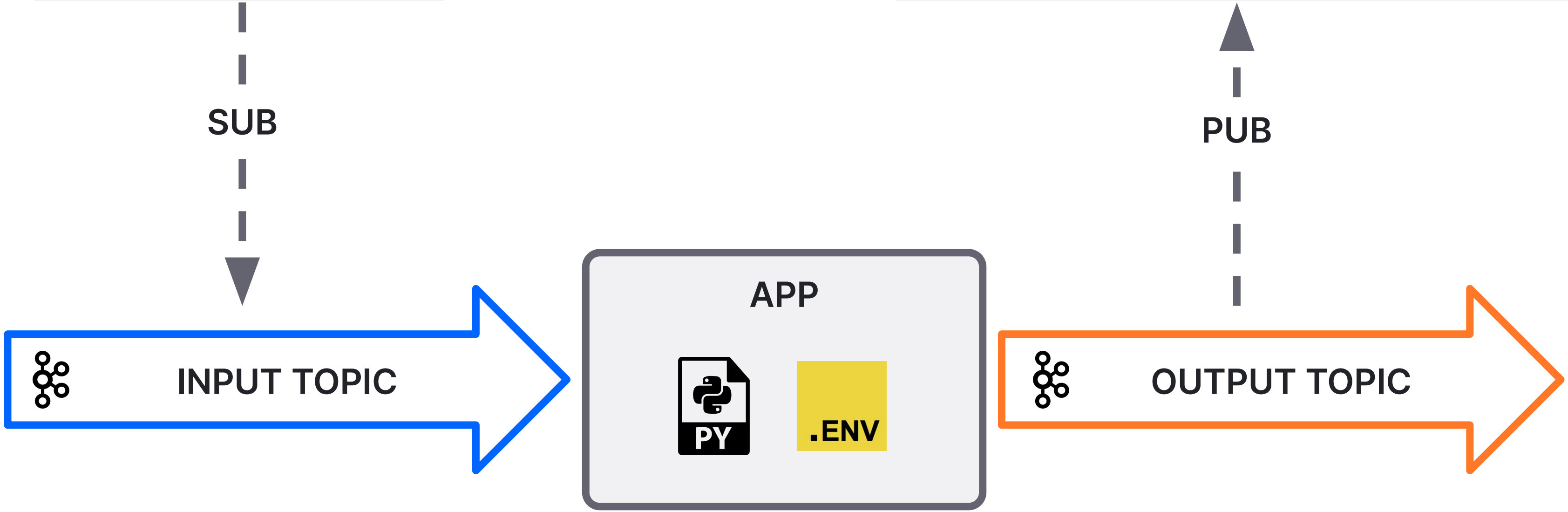
Python

Python gives you flexibility. It lets you transform data, not just query it. From simple filtering to ML use cases like video processing.

Processing with streaming

gForce X	gForce Y	gForce Z
0.5	0.3	0.1

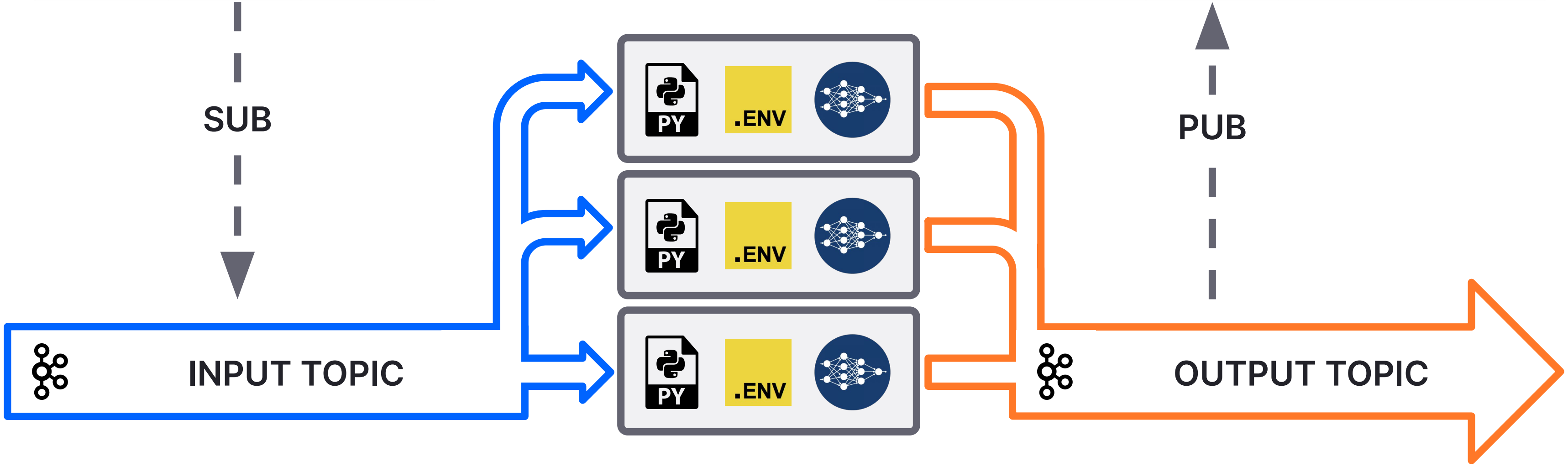
gForce X	gForce Y	gForce Z	gForce Total	Crash
0.5	0.3	0.1	0.9	1



Scale

gForce X	gForce Y	gForce Z
0.5	0.3	0.1

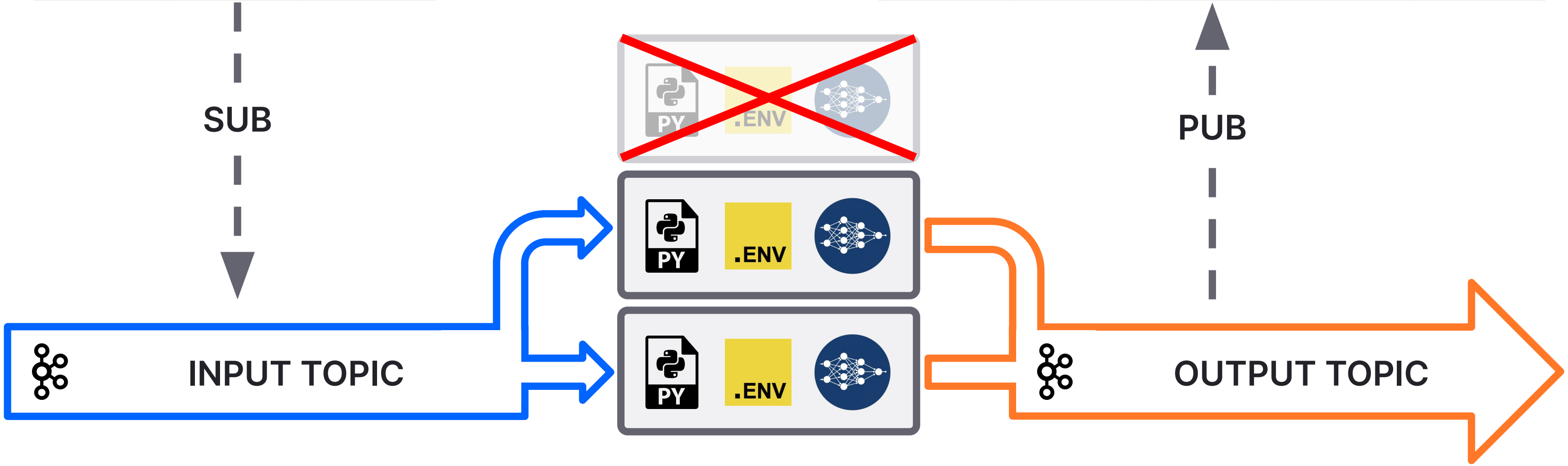
gForce X	gForce Y	gForce Z	gForce Total	Crash
0.5	0.3	0.1	0.9	1



Fault tolerant

gForce X	gForce Y	gForce Z
0.5	0.3	0.1

gForce X	gForce Y	gForce Z	gForce Total	Crash
0.5	0.3	0.1	0.9	1



Try Quix

Sign up

GitHub

Thank you

Quix

info@quix.io | www.quix.io

InfluxDB Platform

Database & platform for handling time series data at massive scale