influxdata®

Quix

Saving the Holidays with Quix and InfluxDB: The OpenTelemetry Anomaly Detection Story

December 2023

Quix | influxdata®

Tun Shwe

VP of Data, Quix

Past life: Head of Data and Data Engineer.

Passion: Event-driven/real-time/streaming technologies and all things audio/music.

Driven: To create a new normal where data is processed as soon as it is generated.

Belief: Less is more. Get started sooner.

Jay Clifford

Developer Advocate, InfluxData

Past life: Sales Engineer for IIoT Solutions.

Passion: Apache Ecosystem, Big Data and Demo tinkering.

Driven: To make observability and IoT solutions accessible to all.

Belief: An industry’s success comes from the domain experts.

Agenda

1

What really is data plumbing?

How does this relate to event streaming and TSDBs

2

Let's look at a problem

Problems drive learning.
Let’s create a scenario with a problem to solve.

3

Let’s solve that problem

We will deploy Quix (streaming platform) and InfluxDB (TSDB) to solve the problem

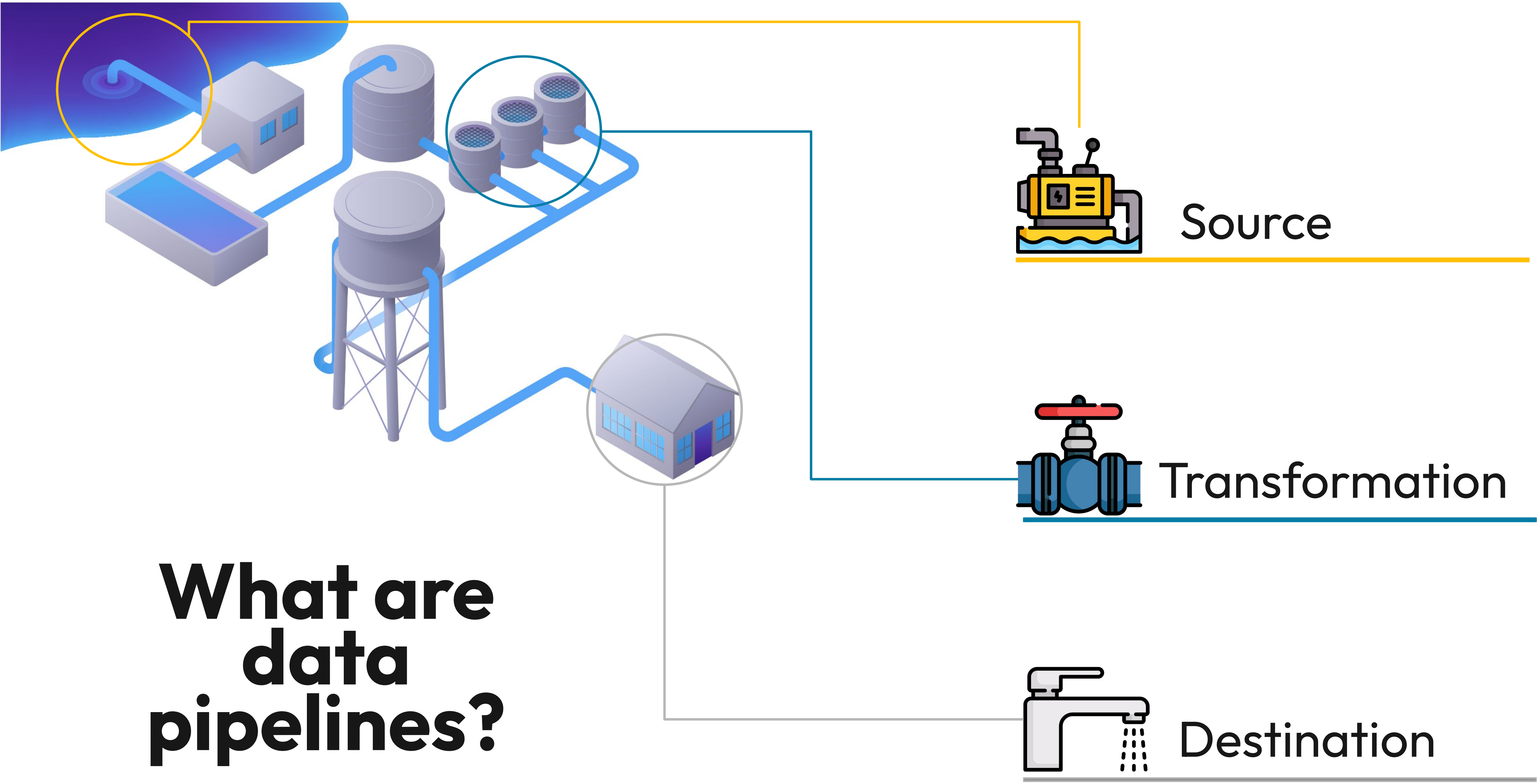
An intro to OTEL

As our data pipeline grows what can be do to make sure that we know what’s happening at each stage/

4

Next steps

Get your hands on the source code and get involved with our communities.



Source

Transformation

Destination

What are data pipelines?

Introduction to event streaming and TSDBs

Streaming technologies

APACHE kafka

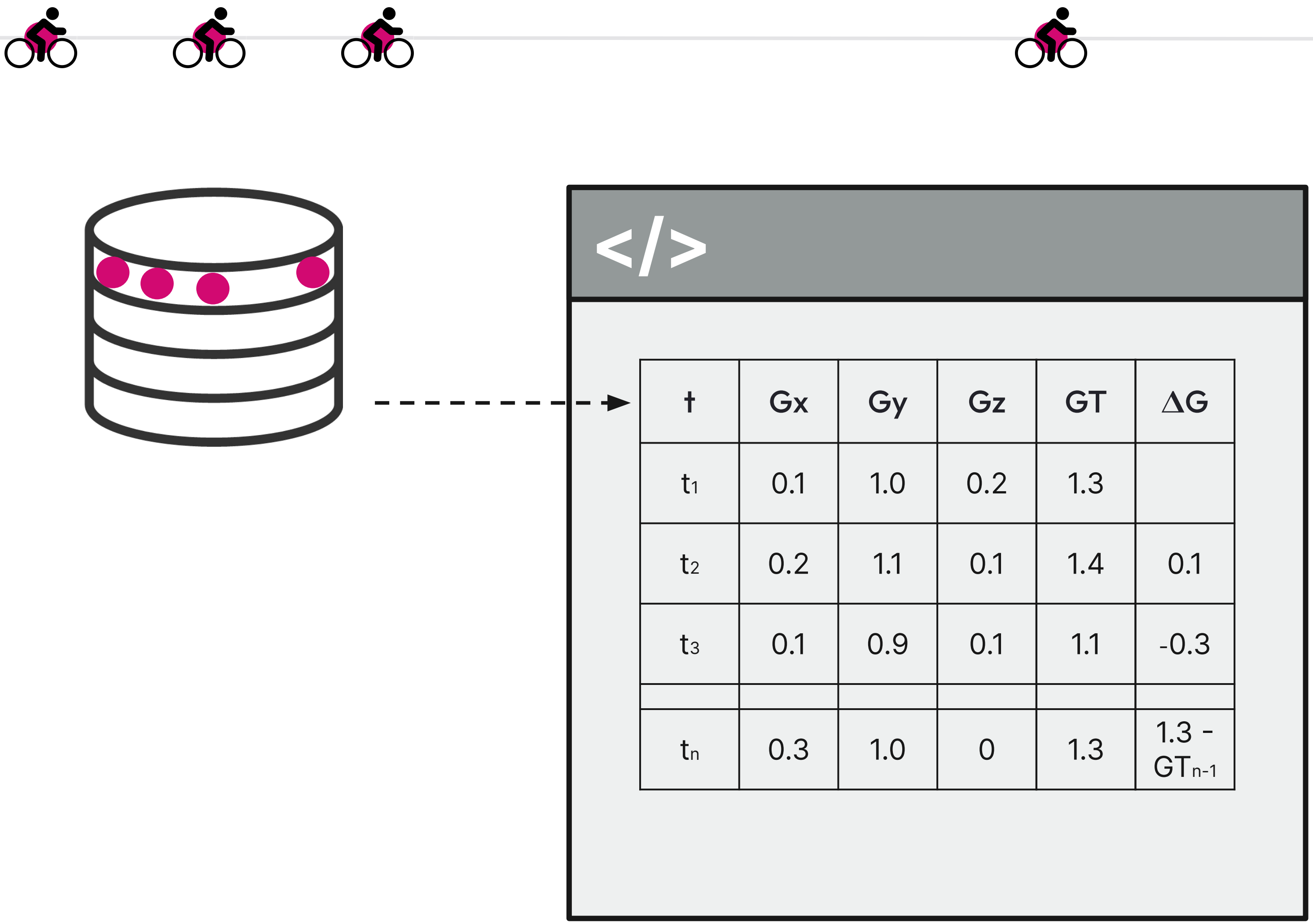
PULSAR

Batching

Data is collected in a database.

Bounded data is periodically scheduled to be loaded from the database into the processing system.

- Computation on range of historical data (stateless).
- Process data at rest.
- Results are not in real time.



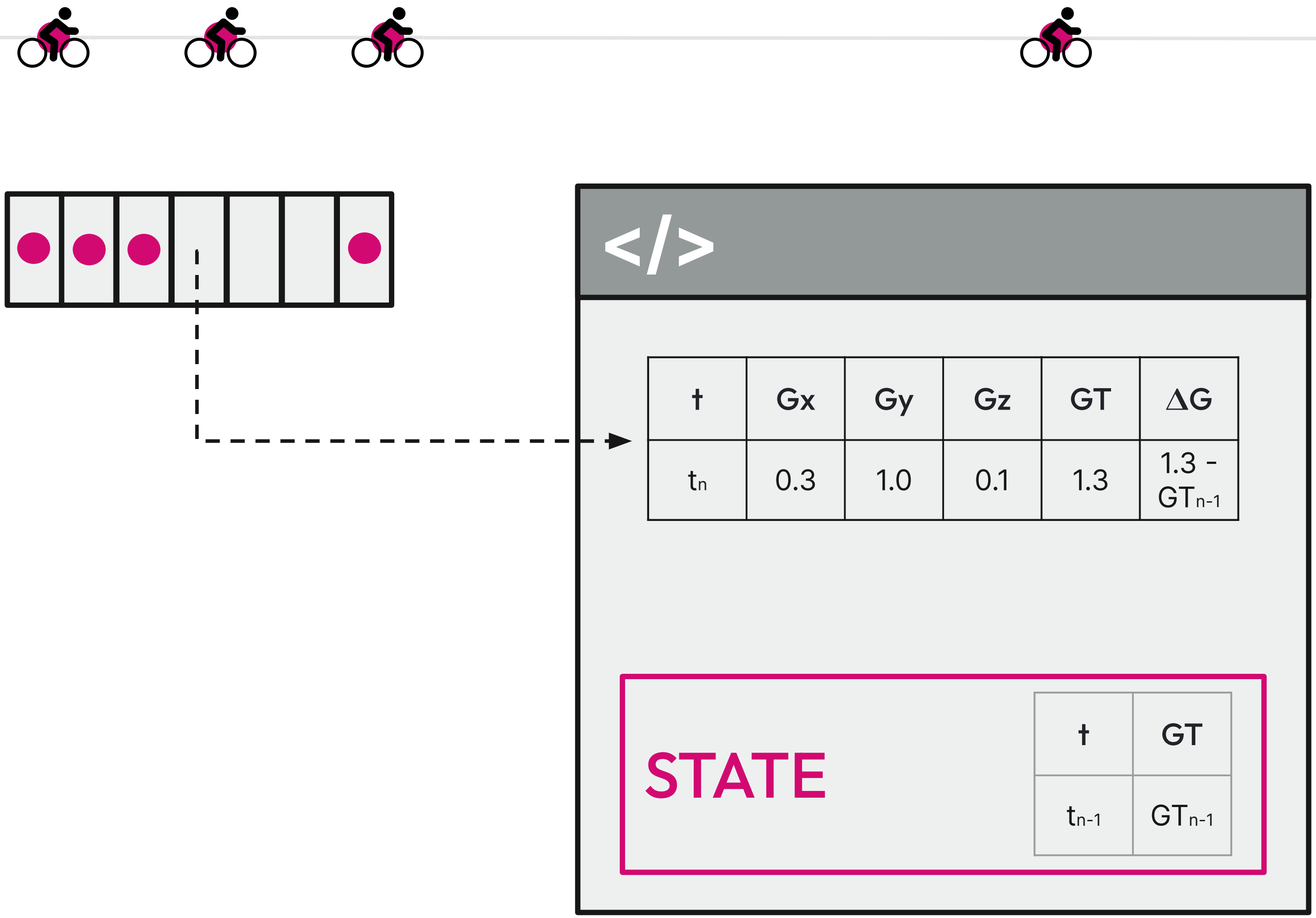
t	Gx	Gy	Gz	GT	ΔG
t_1	0.1	1.0	0.2	1.3	
t_2	0.2	1.1	0.1	1.4	0.1
t_3	0.1	0.9	0.1	1.1	-0.3
t_n	0.3	1.0	0	1.3	$1.3 - GT_{n-1}$

Streaming

Data is collected in a broker or transport, e.g. a Kafka topic.

Unbounded data is continuously consumed and processed as soon as it is published to the topic.

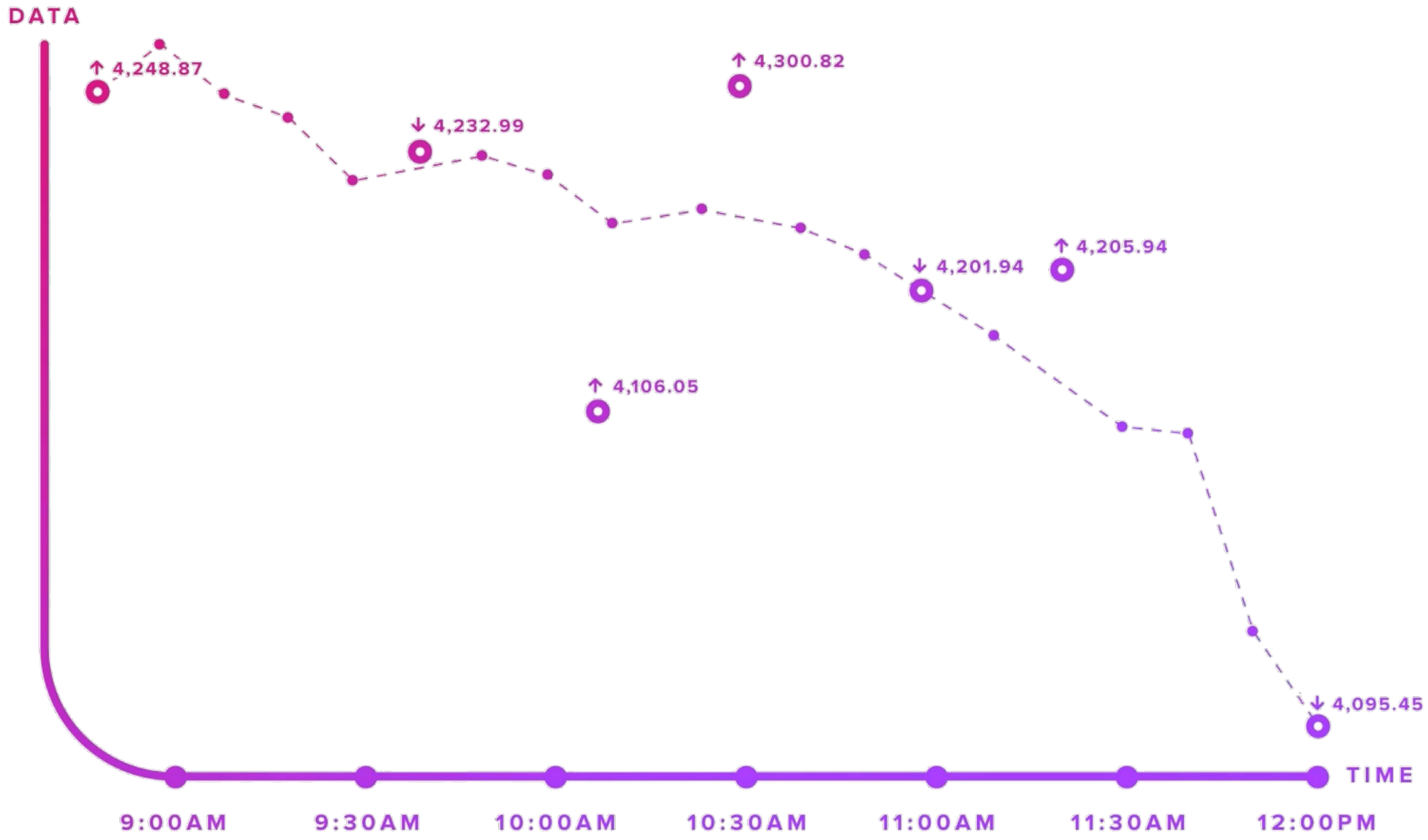
- Computation on each event (stateful where necessary).
- Process data in motion.
- Real-time results.



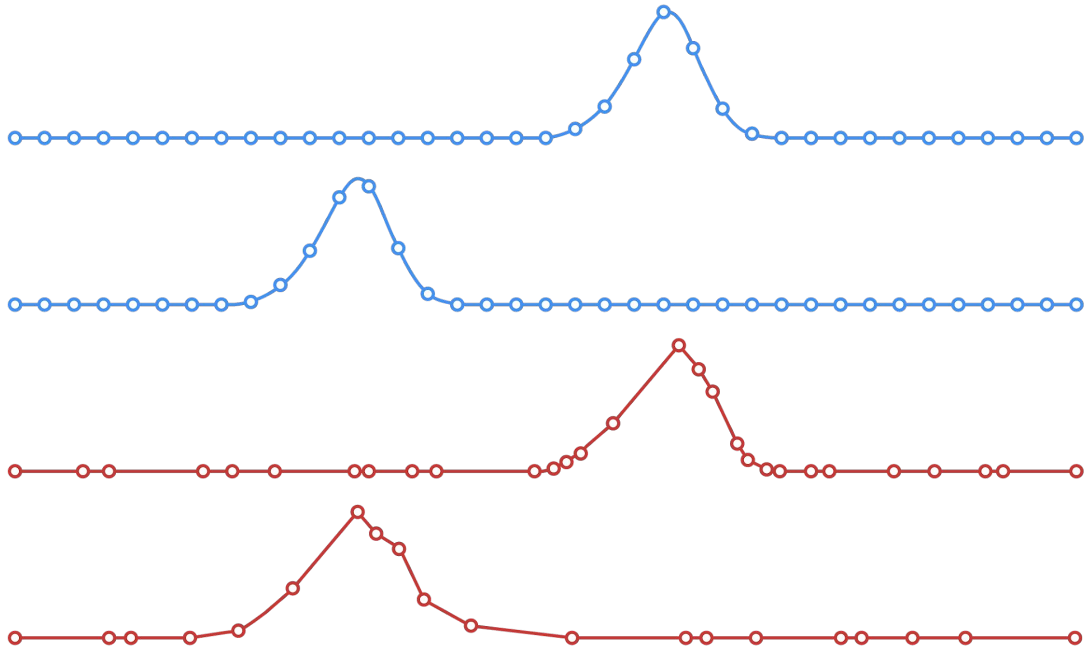
t	Gx	Gy	Gz	GT	ΔG
t_n	0.3	1.0	0.1	1.3	1.3 - GT_{n-1}

t	GT
t_{n-1}	GT_{n-1}

Time series data



Types of time series



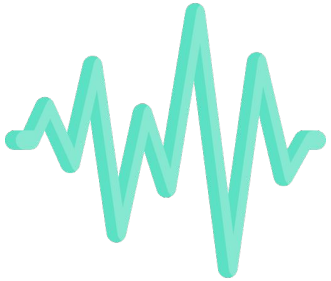
Metrics

Measurements at **regular** time intervals

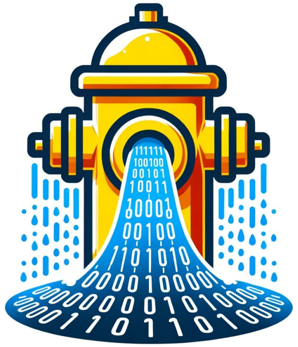
Events

Measurements at **irregular** time intervals

What is a time series database?

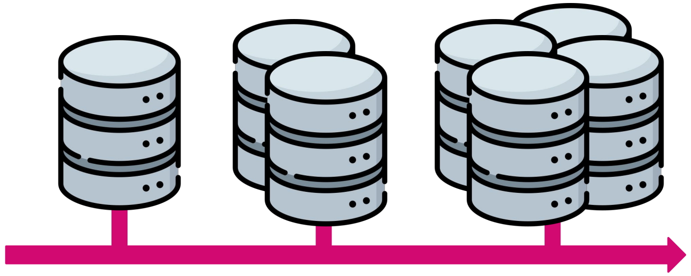


Time series data



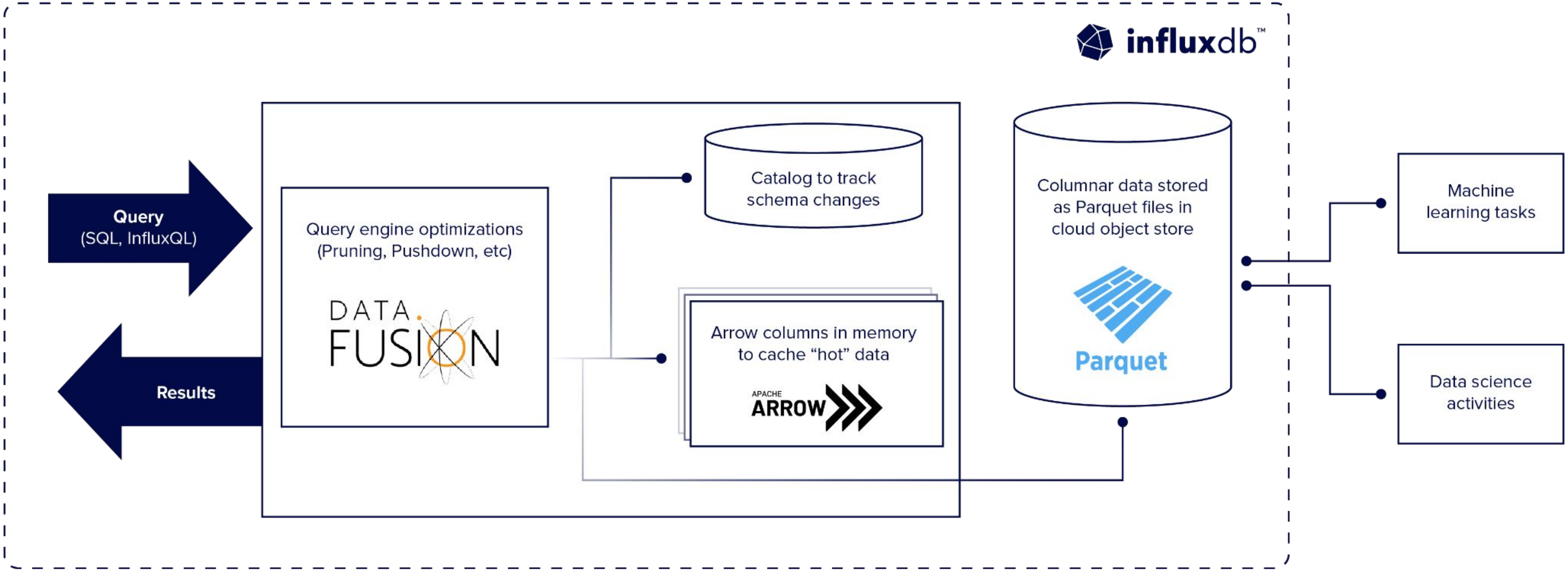
High write throughput

Efficient queries over time ranges



Scalability and performance

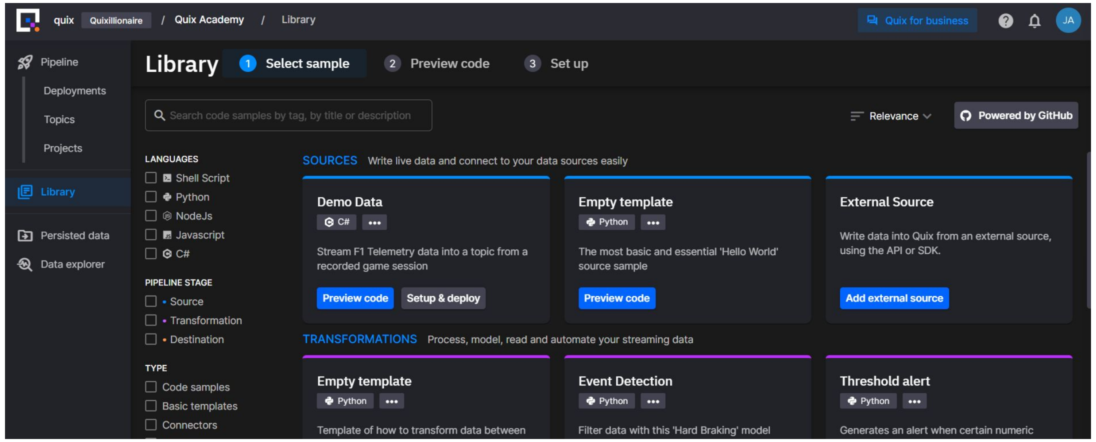
InfluxDB 3.0


influxdb™
Query
(SQL, InfluxQL)
Results
Query engine optimizations
(Pruning, Pushdown, etc)
DATA FUSION
Catalog to track
schema changes
Arrow columns in memory
to cache "hot" data
APACHE ARROW
Columnar data stored
as Parquet files in
cloud object store
Parquet
Machine
learning tasks
Data science
activities

Quix ✌️

● Quix Streams

Open source library for processing data in Kafka using pure Python. Inspired by FaaS and pandas



quix Quixillionaire / Quix Academy / Library

Quix for business

JA

Pipeline
Deployments
Topics
Projects
Library
Persisted data
Data explorer

Library ① Select sample ② Preview code ③ Set up

Search code samples by tag, by title or description

Relevance

Powered by GitHub

LANGUAGES
- □ Shell Script
- □ Python
- □ NodeJs
- □ Javascript
- □ C#

PIPELINE STAGE
- □ Source
- □ Transformation
- □ Destination

TYPE
- □ Code samples
- □ Basic templates
- □ Connectors

SOURCES Write live data and connect to your data sources easily

Demo Data
C# •••
Stream F1 Telemetry data into a topic from a recorded game session
Preview code | Setup & deploy

Empty template
Python •••
The most basic and essential 'Hello World' source sample
Preview code

External Source
Write data into Quix from an external source, using the API or SDK.
Add external source

TRANSFORMATIONS Process, model, read and automate your streaming data

Empty template
Python •••
Template of how to transform data between

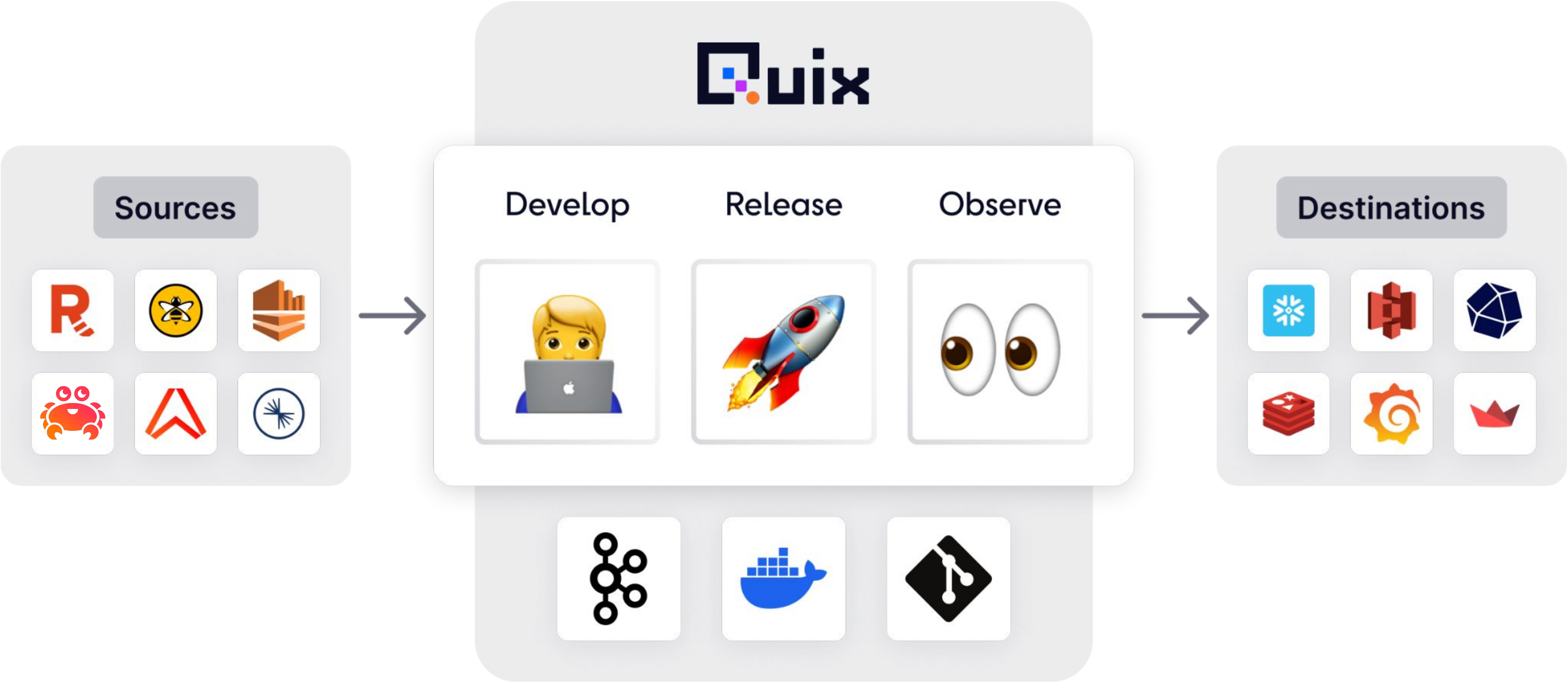
Event Detection
Python •••
Filter data with this 'Hard Braking' model

Threshold alert
Python •••
Generates an alert when certain numeric

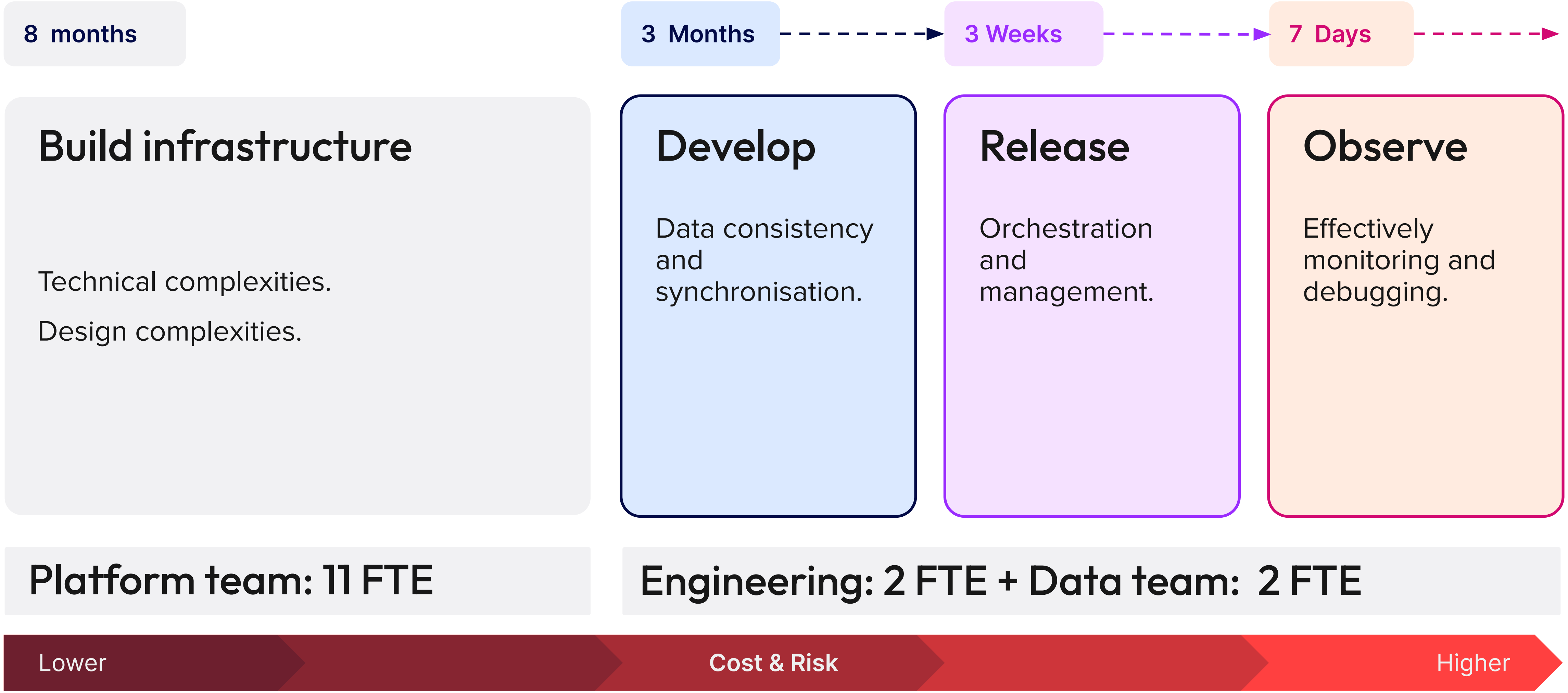
■ Quix Cloud

Platform to quickly build, test and deploy streaming data pipelines and applications without having to manage infrastructure

Quix platform architecture



Building your own architecture is costly



8 months

3 Months → 3 Weeks → 7 Days →

Build infrastructure

Technical complexities.

Design complexities.

Develop

Data consistency and synchronisation.

Release

Orchestration and management.

Observe

Effectively monitoring and debugging.

Platform team: 11 FTE

Engineering: 2 FTE + Data team: 2 FTE

Quix | influxdata®

Accelerated application development

Weeks

Develop

Use free open source connectors & code samples to get started. Enable ML and GenAI faster.

Hours

Release

IaC: code, test and deploy event streaming applications. Powered by Kafka, Docker and Git.

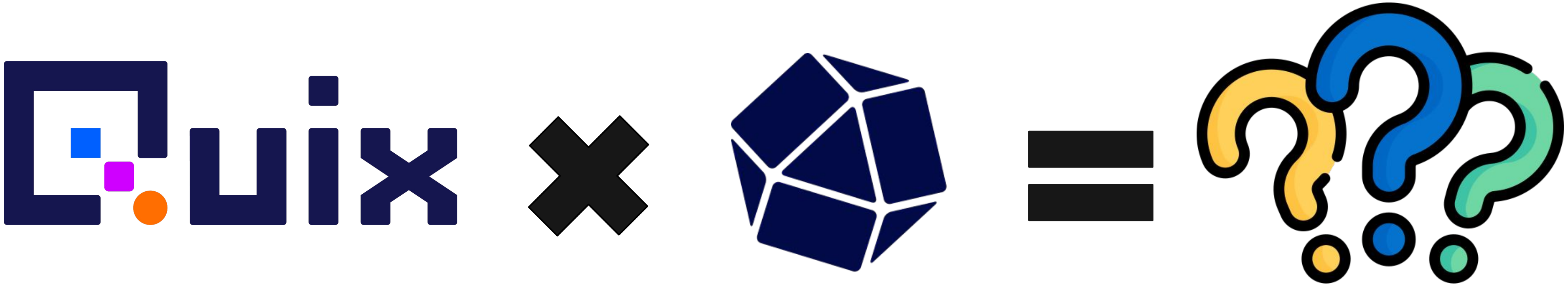
Minutes

Observe

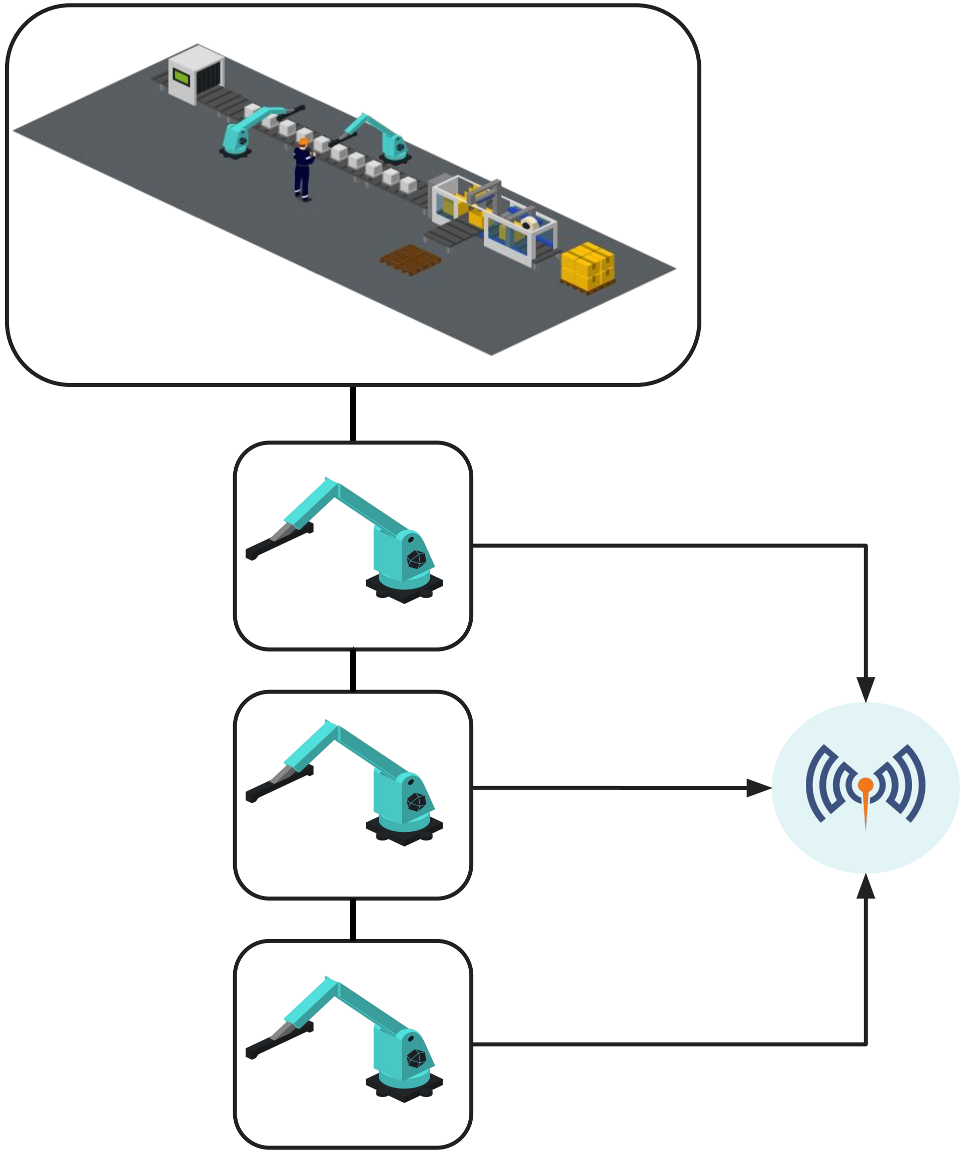
A suite of observability tools for in-depth insights into your event-driven architecture.

Engineering: 2 FTE + Data team: 2 FTE

Predictable Cost & Risk


uix

Let's look at a problem



📦 Packing Co — Anomaly Detection

😥 Packing Co is having **recurring issues** with their packaging machines.

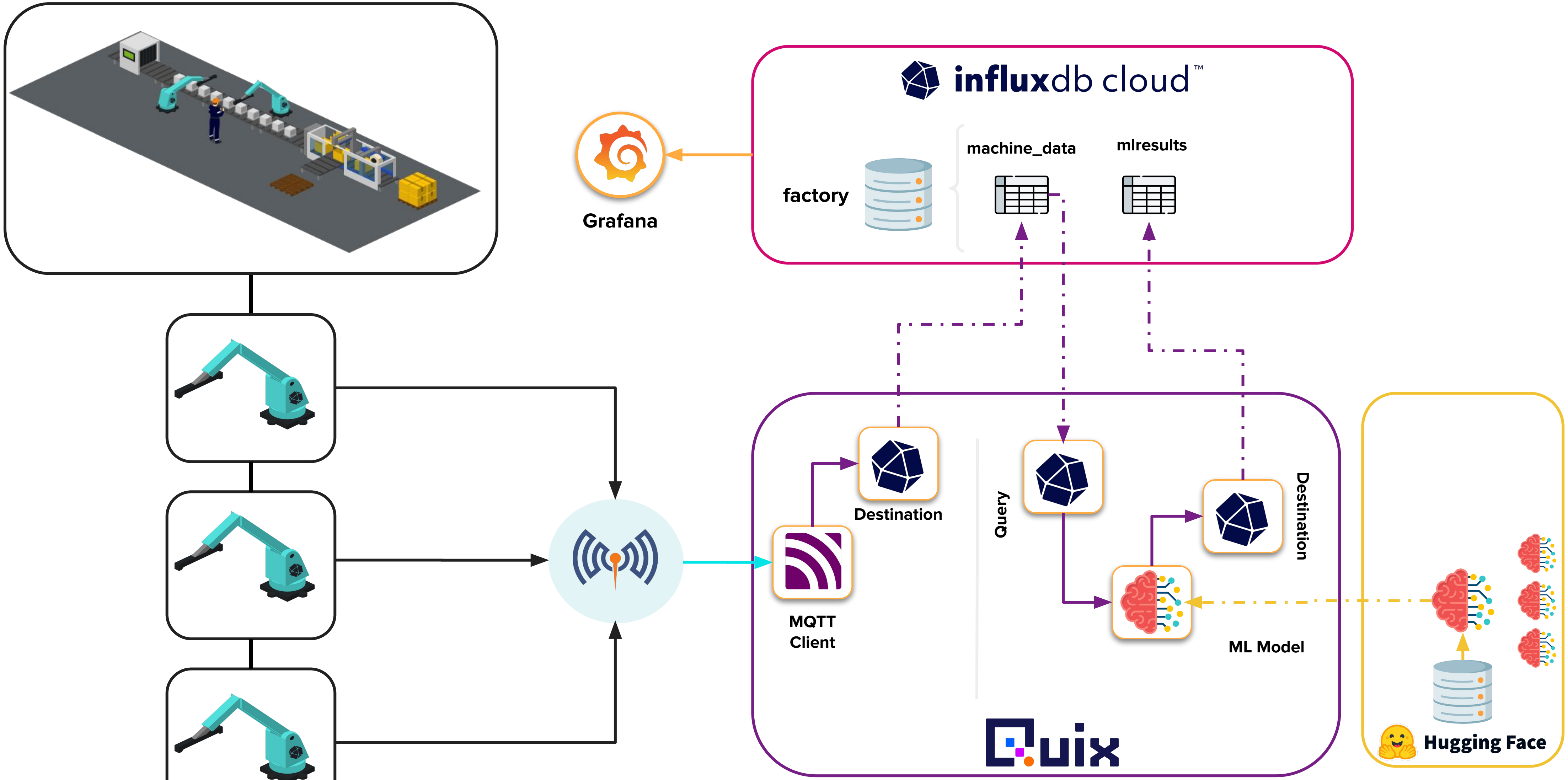
🤖 Unexpectedly, any of the machines will enter a **failing state**, which requires a manual reset by an engineer.

📊 The Plant Manager has advised, **when running normally** all machine sensors will follow **similar output patterns**. If a machine is at **fault** these will **fluctuate abnormally.**

🤔 How can we use **Quix** and **InfluxDB** to solve this?

Let’s solve that problem

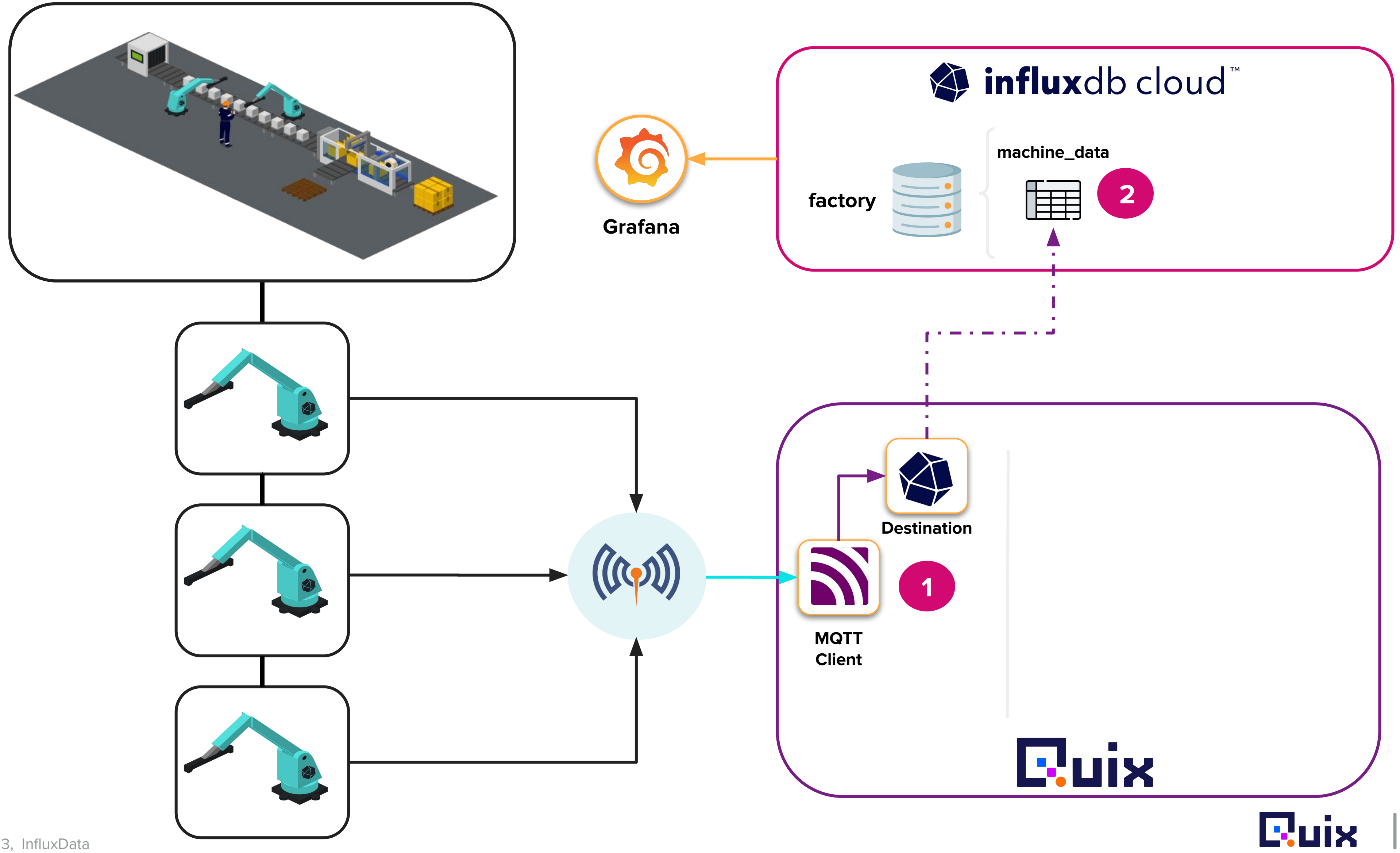
Solution Architecture



Quix | influxdata®

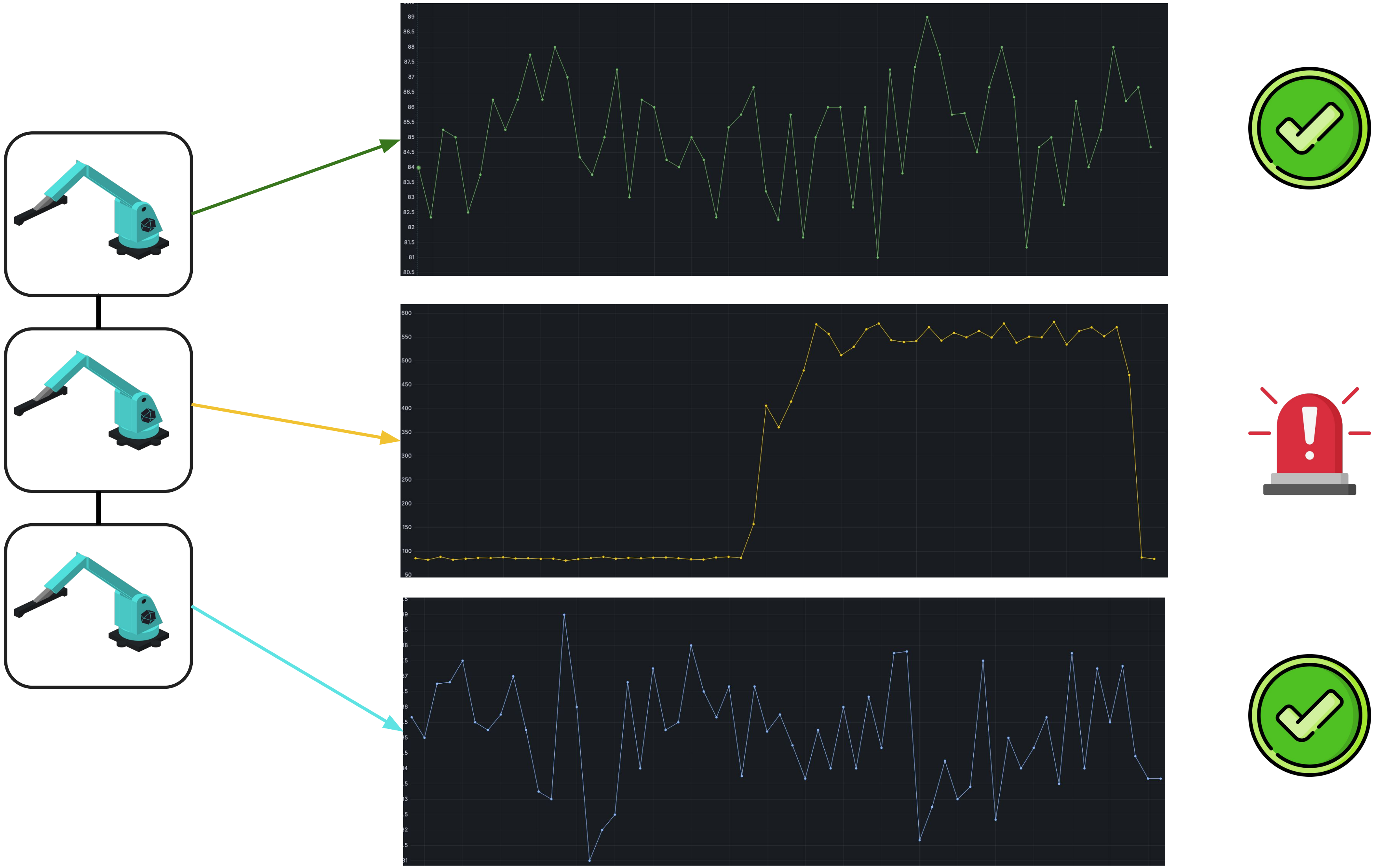
Data Ingest

Solution Architecture (Ingest)



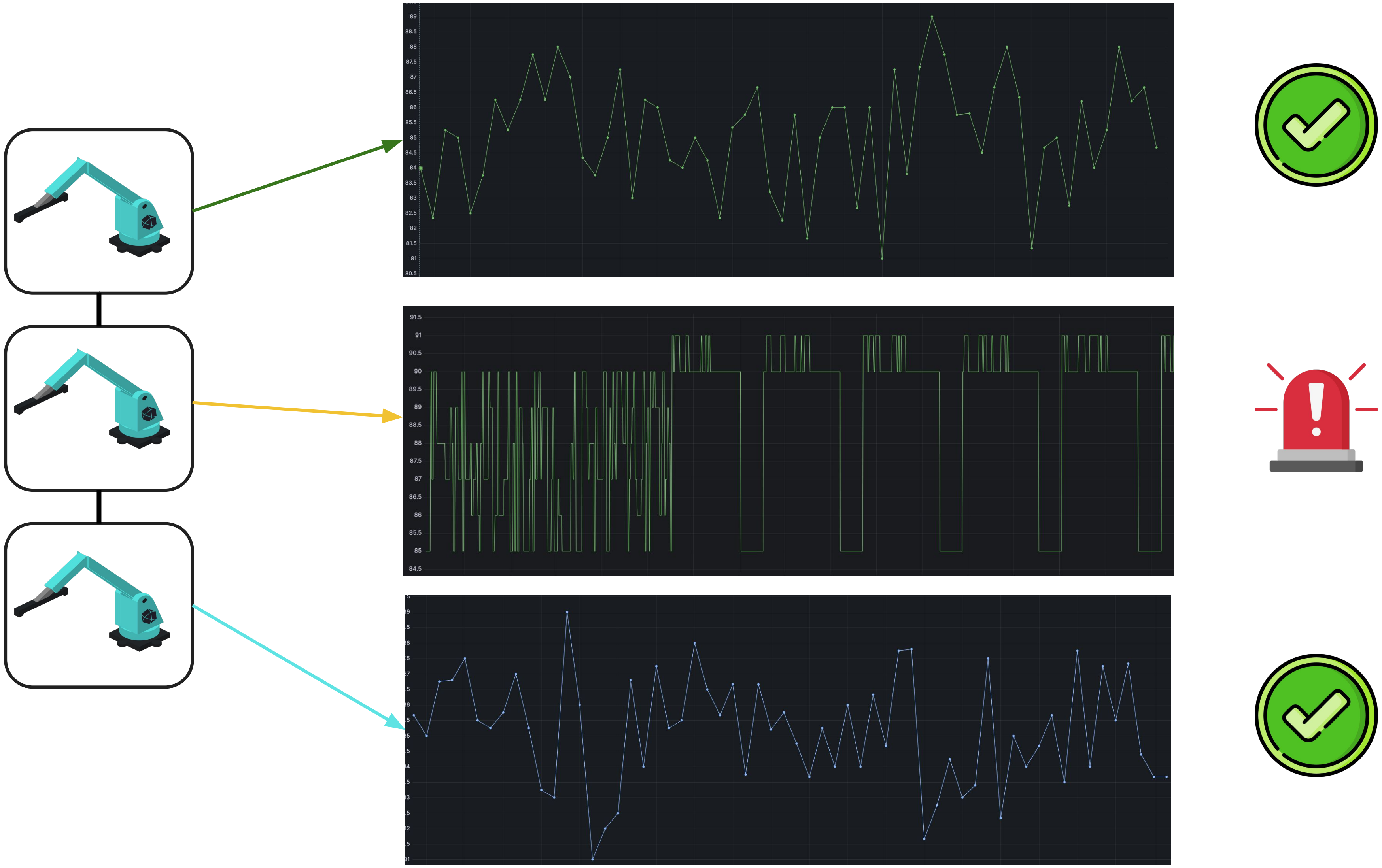
Choosing a Model

“Less is more” - Tun Shwe



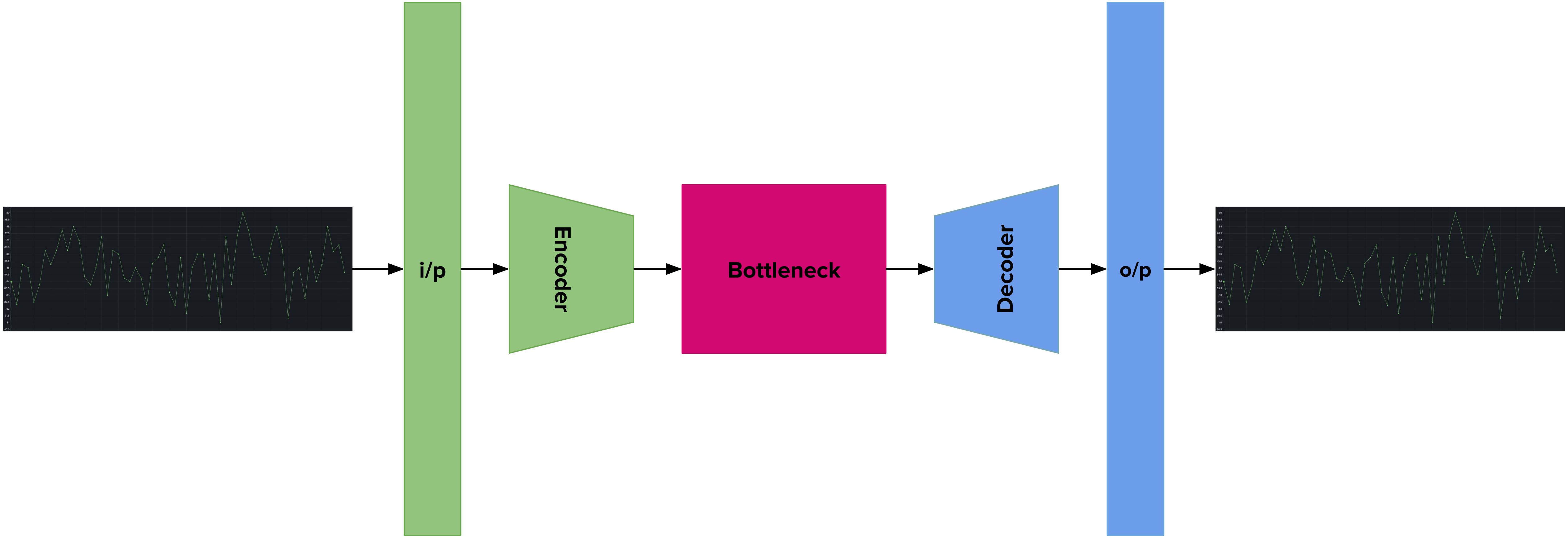
This could easily be solved with thresholding

“Success comes from the domain experts” - Jay Clifford



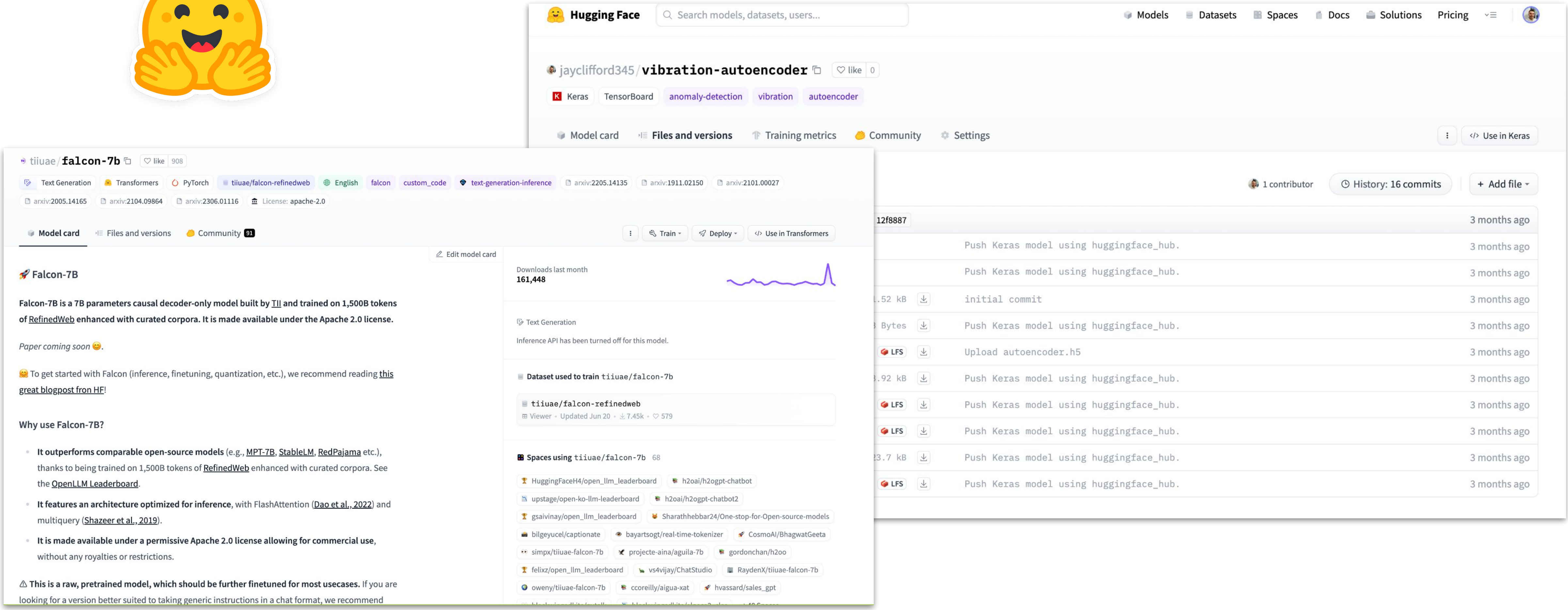
What do we do when our result becomes unpredictable by conventional means?

Machine learning – Autoencoder



ML Deployment

Hugging Face



jayclifford345/**vibration-autoencoder** like 0

Keras · TensorBoard · anomaly-detection · vibration · autoencoder

```
12f8887                                                    3 months ago
                    Push Keras model using huggingface_hub.   3 months ago
                    Push Keras model using huggingface_hub.   3 months ago
.52 kB              initial commit                            3 months ago
 Bytes              Push Keras model using huggingface_hub.   3 months ago
LFS                 Upload autoencoder.h5                     3 months ago
.92 kB              Push Keras model using huggingface_hub.   3 months ago
LFS                 Push Keras model using huggingface_hub.   3 months ago
LFS                 Push Keras model using huggingface_hub.   3 months ago
23.7 kB             Push Keras model using huggingface_hub.   3 months ago
LFS                 Push Keras model using huggingface_hub.   3 months ago
```

tiiuae/**falcon-7b** like 908

Text Generation · Transformers · PyTorch · tiiuae/falcon-refinedweb · English · falcon · custom_code · text-generation-inference · arxiv:2205.14135 · arxiv:1911.02150 · arxiv:2101.00027 · arxiv:2005.14165 · arxiv:2104.09864 · arxiv:2306.01116 · License: apache-2.0

🚀 Falcon-7B

Falcon-7B is a 7B parameters causal decoder-only model built by TII and trained on 1,500B tokens of RefinedWeb enhanced with curated corpora. It is made available under the Apache 2.0 license.

Paper coming soon 😊.

🤗 To get started with Falcon (inference, finetuning, quantization, etc.), we recommend reading this great blogpost fron HF!

Why use Falcon-7B?

- **It outperforms comparable open-source models** (e.g., MPT-7B, StableLM, RedPajama etc.), thanks to being trained on 1,500B tokens of RefinedWeb enhanced with curated corpora. See the OpenLLM Leaderboard.
- **It features an architecture optimized for inference**, with FlashAttention (Dao et al., 2022) and multiquery (Shazeer et al., 2019).
- **It is made available under a permissive Apache 2.0 license allowing for commercial use**, without any royalties or restrictions.

⚠ **This is a raw, pretrained model, which should be further finetuned for most usecases.** If you are looking for a version better suited to taking generic instructions in a chat format, we recommend

Downloads last month
161,448

Text Generation

Inference API has been turned off for this model.

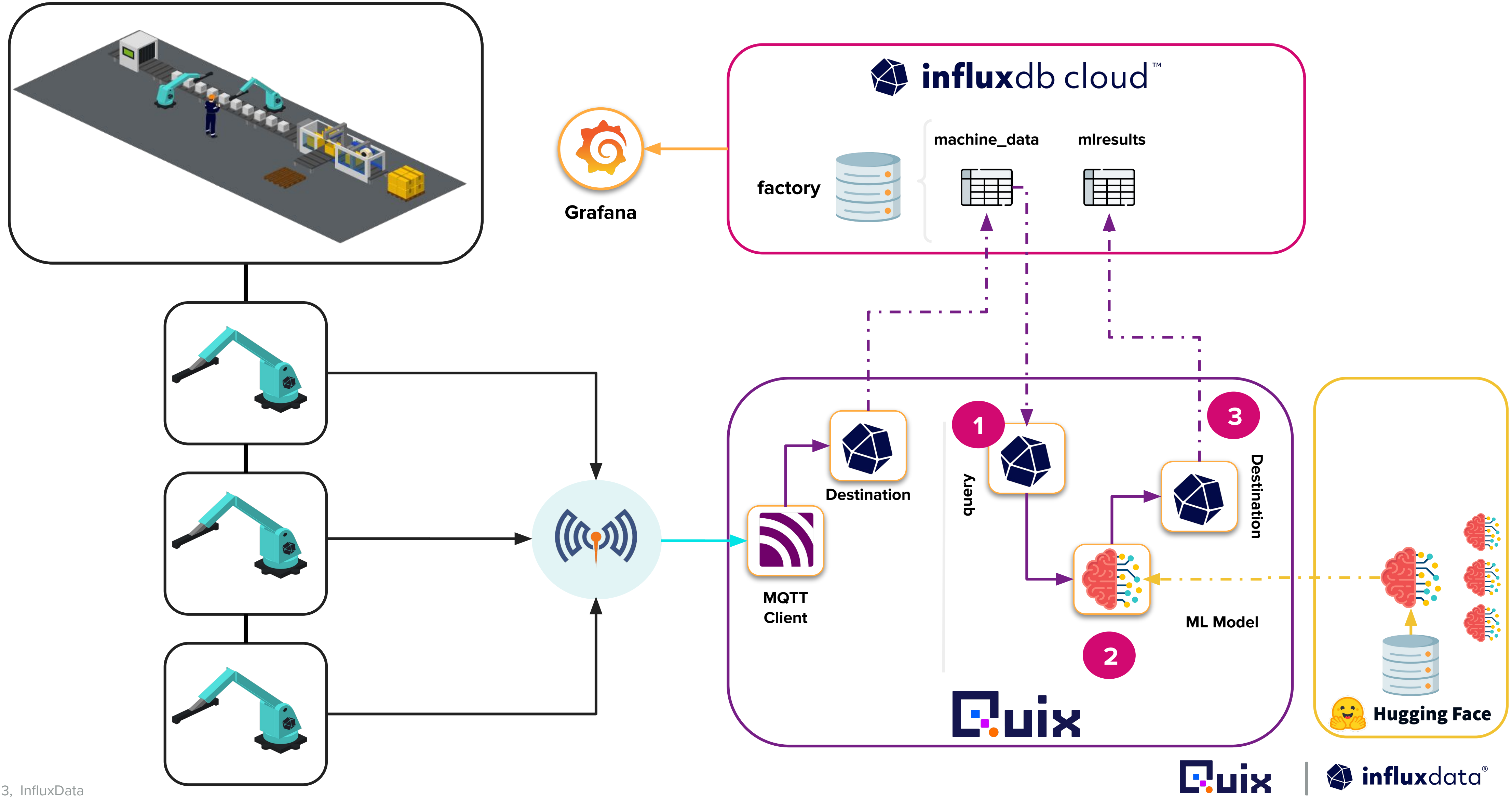
Dataset used to train tiiuae/falcon-7b

tiiuae/falcon-refinedweb
Viewer • Updated Jun 20 • 7.45k • ♡ 579

Spaces using tiiuae/falcon-7b 68

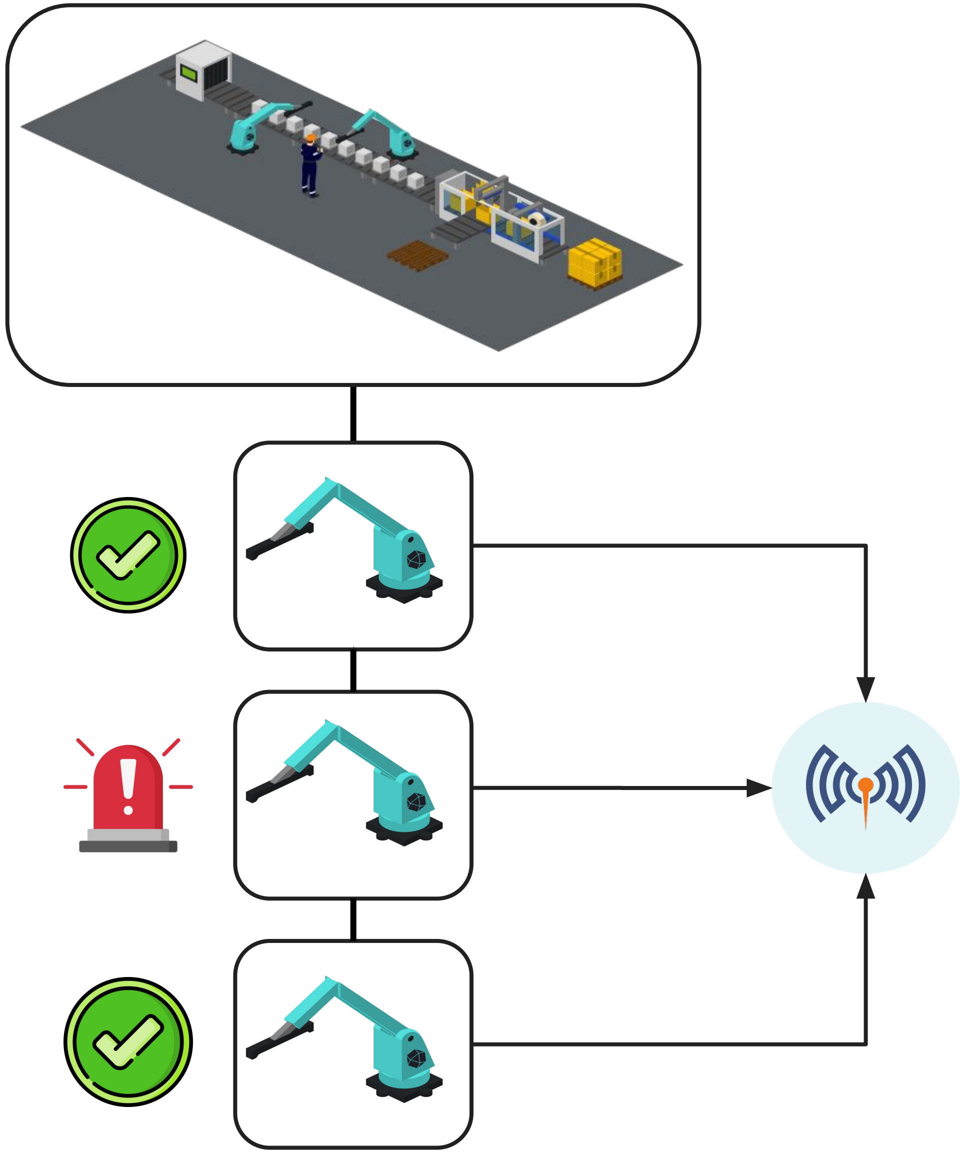
HuggingFaceH4/open_llm_leaderboard · h2oai/h2ogpt-chatbot · upstage/open-ko-llm-leaderboard · h2oai/h2ogpt-chatbot2 · gsaivinay/open_llm_leaderboard · Sharathhebbar24/One-stop-for-Open-source-models · bilgeyucel/captionate · bayartsogt/real-time-tokenizer · CosmoAI/BhagwatGeeta · simpx/tiiuae-falcon-7b · projecte-aina/aguila-7b · gordonchan/h2oo · felixz/open_llm_leaderboard · vs4vijay/ChatStudio · RaydenX/tiiuae-falcon-7b · oweny/tiiuae-falcon-7b · ccoreilly/aigua-xat · hvassard/sales_gpt

Solution Architecture (ML Deployment)



Quix | influxdata

So what have solved?



😃 Packing Co — Happy!

✅ Enabled the **ingest, transformation and storage** of their machine data.

✅ Deployed an initial **machine learning model** to detect potential malfunctions using vibration data from the machines.

✅ Provided the foundations of a **scalable data pipeline**.

✅ **Saved the holidays**.

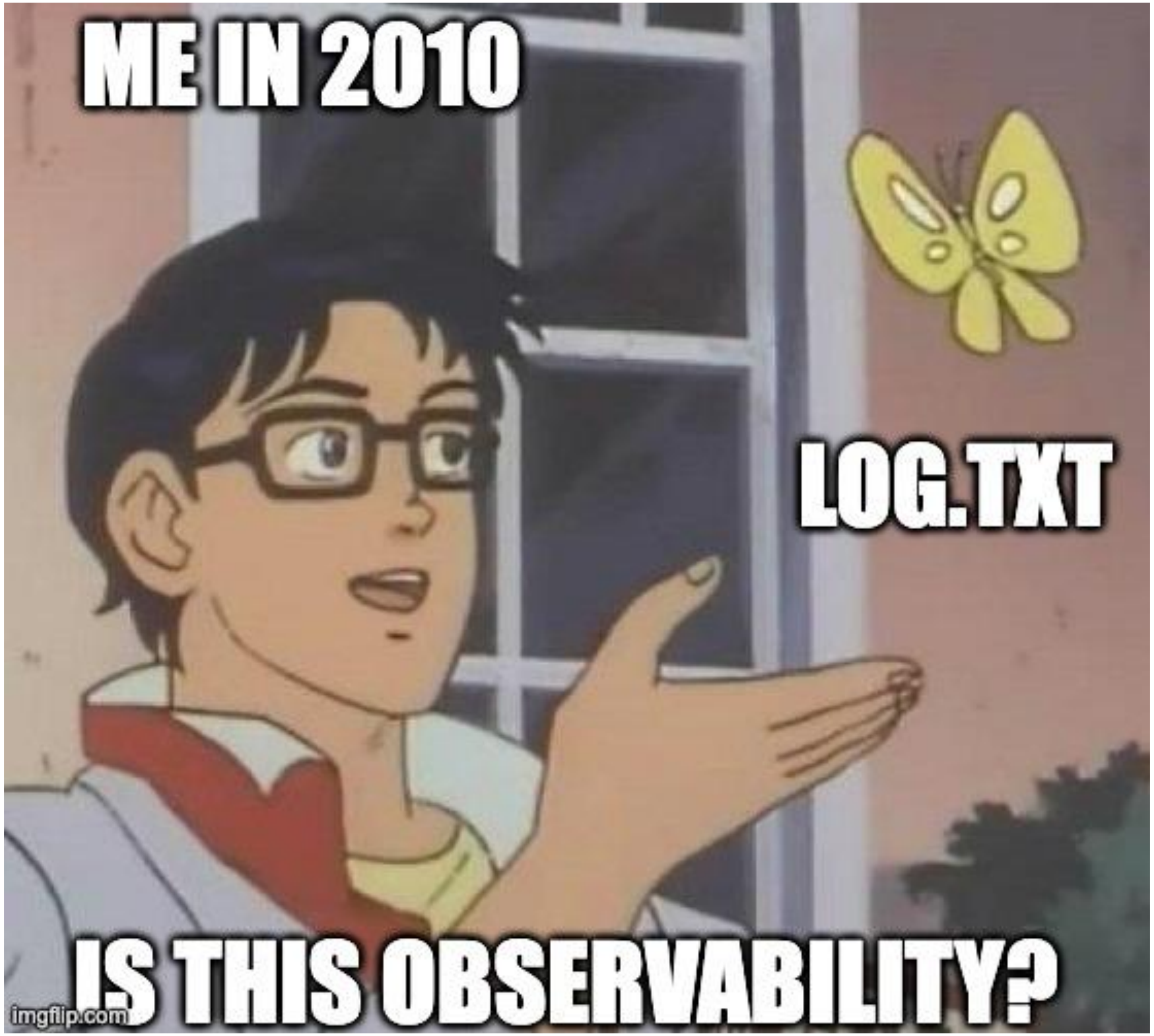
The history of observability

What is observability?

🤓 In control theory, observability is a measure of how well internal states of a system can be inferred from knowledge of its external outputs.

😶🌫️ In distributed systems, observability is the ability to collect data about programs' execution, modules' internal states and the communication among components.

🤩 Observability is a full understanding of our systems.


ME IN 2010
LOG.TXT
IS THIS OBSERVABILITY?
imgflip.com

>_SSH

>_SSH

>_SSH

>_SSH

2015

2015-2017

JAEGER

OPENTRACING

OpenCensus

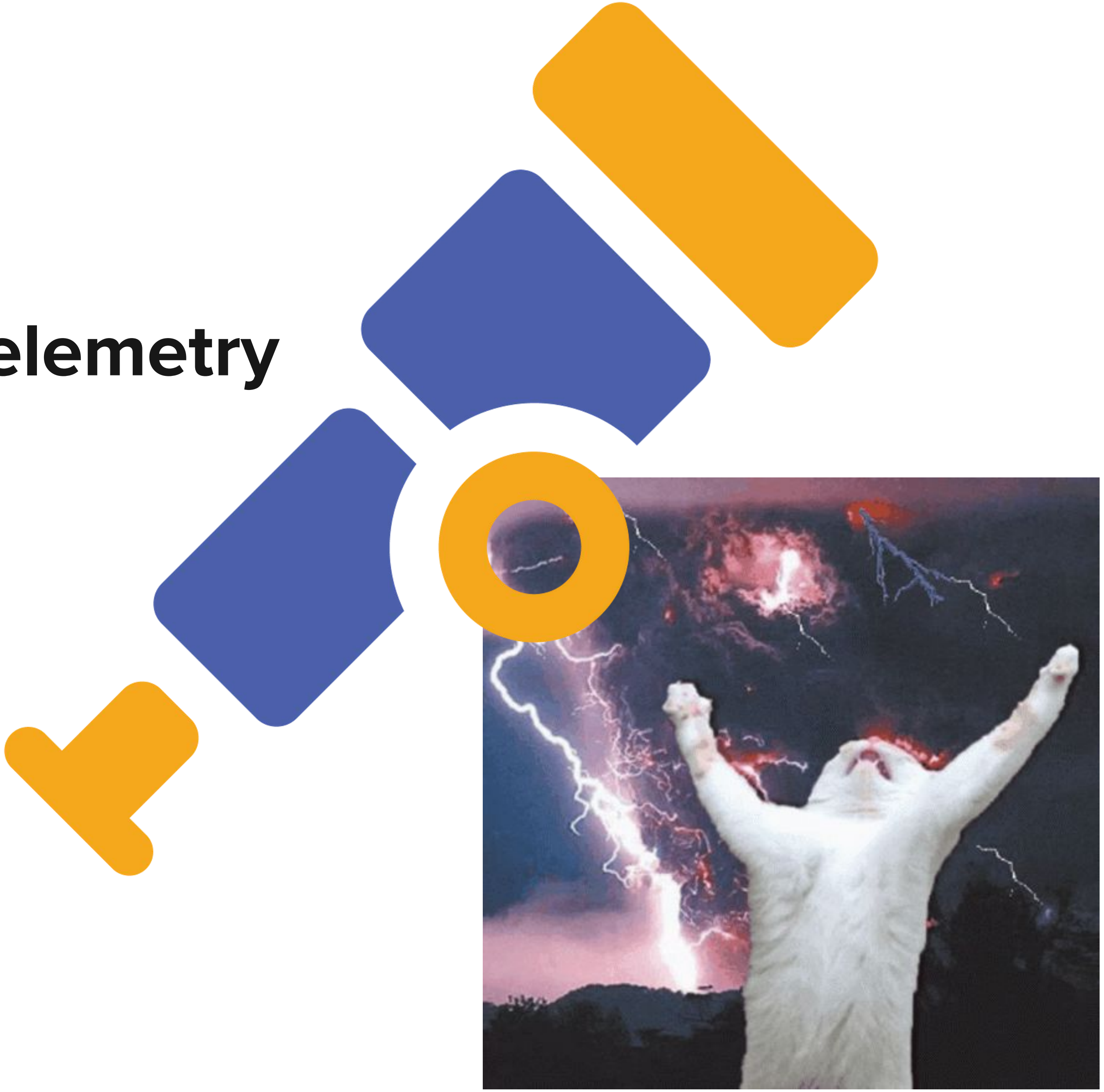
2019

2019

OpenTelemetry

OTel

OTEL



Unification

2022:

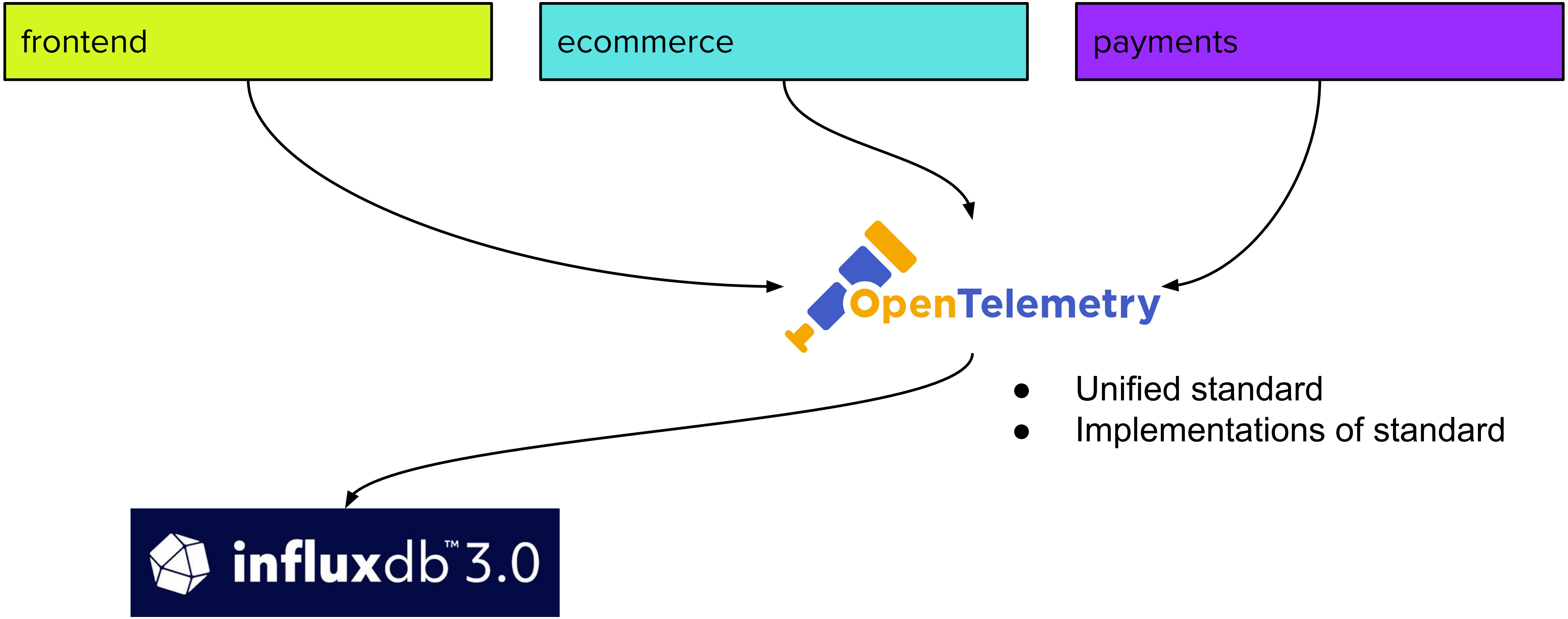
● Logs	push	stdout	logfmt or JSON	Elastic, filesystem
● Metrics	pull	HTTP	Prometheus exposition	InfluxDB, Prometheus
● Traces	push	UDP	Jaeger thrift or gRPC	Elastic, Cassandra

2023:

● Logs	push	gRPC
● Metrics	push	gRPC
● Traces	push	gRPC

OpenTelemetry

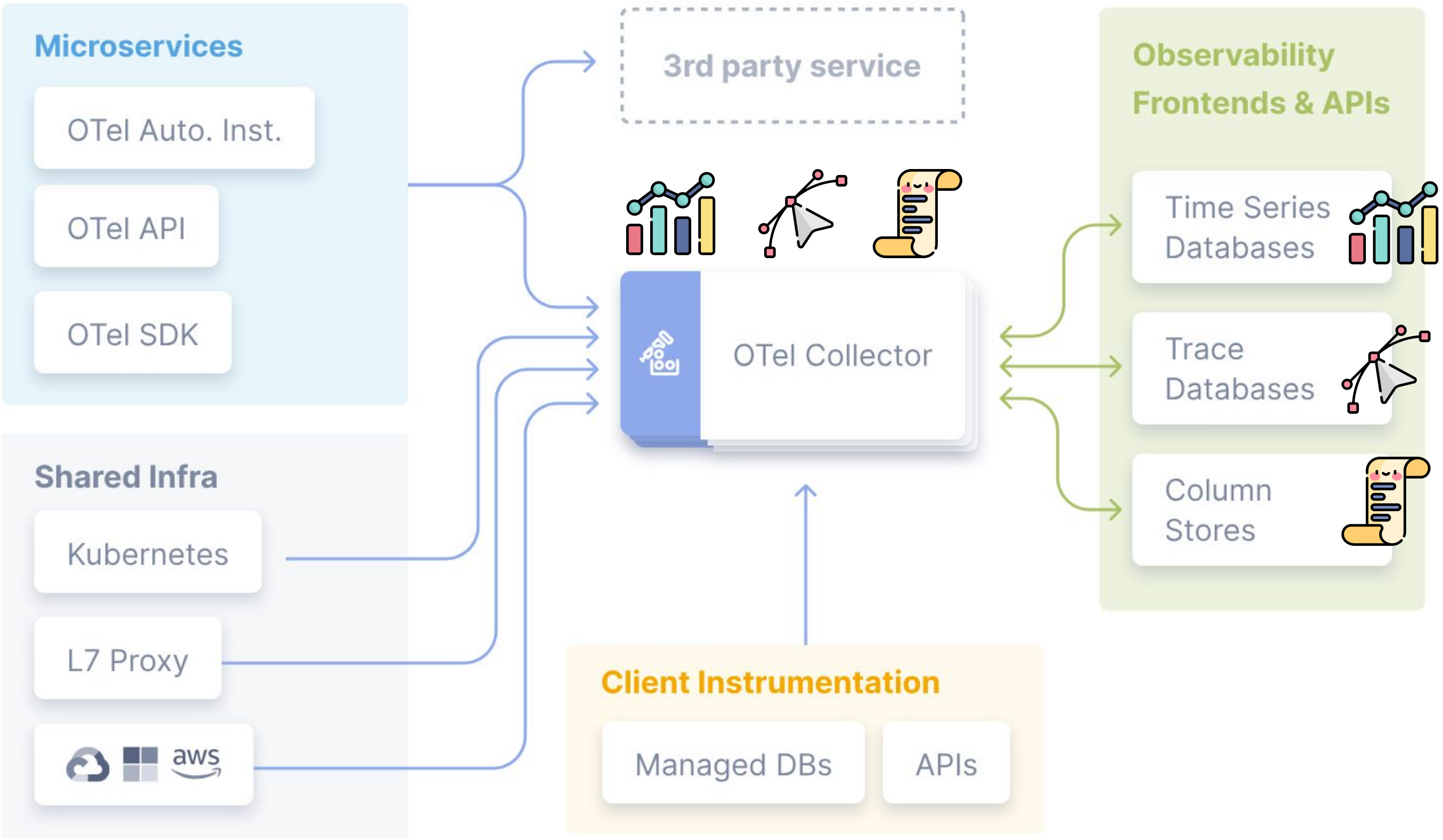
influxdb™ 3.0



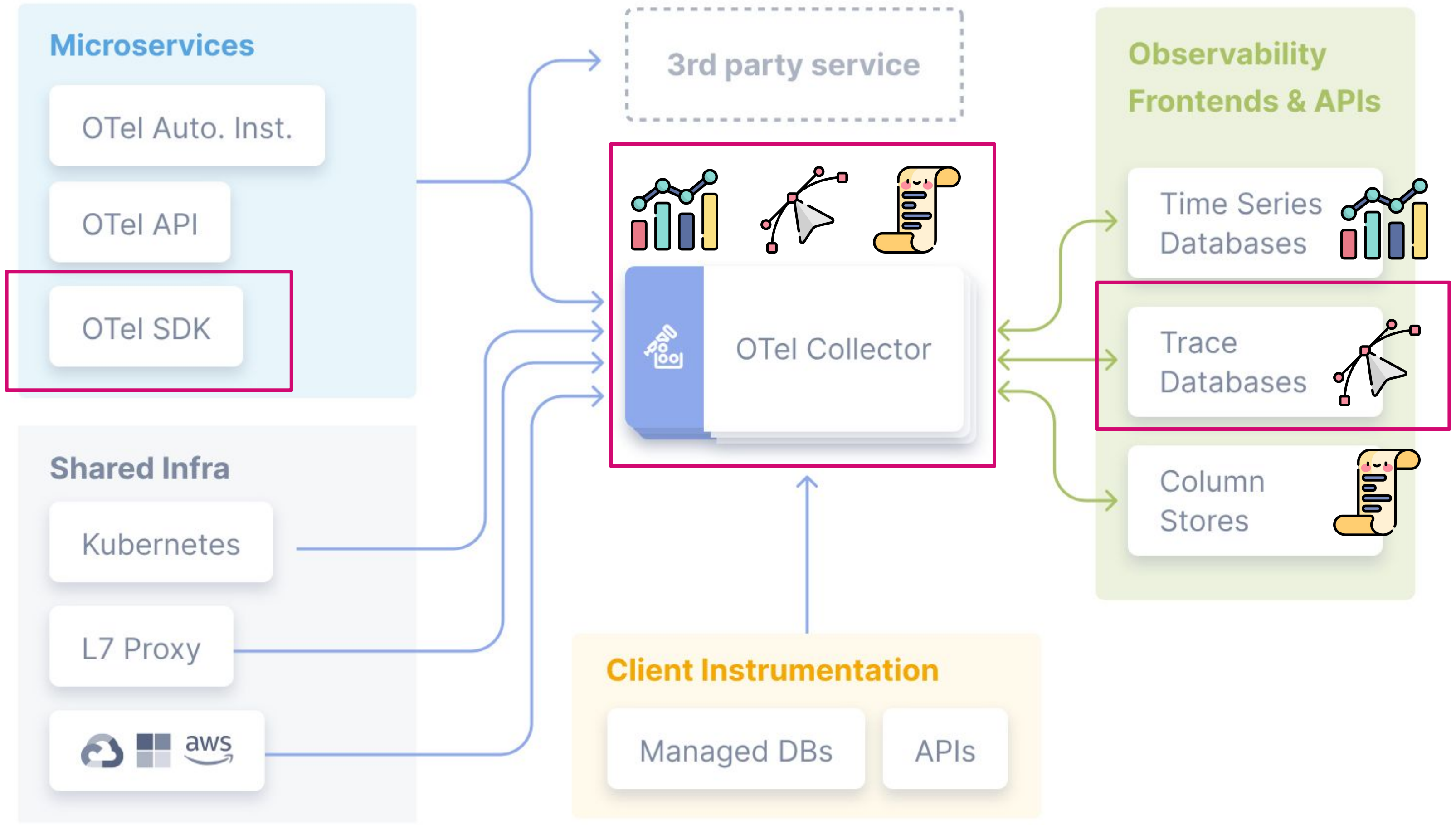
- Unified standard
- Implementations of standard

OpenTelemetry in practice

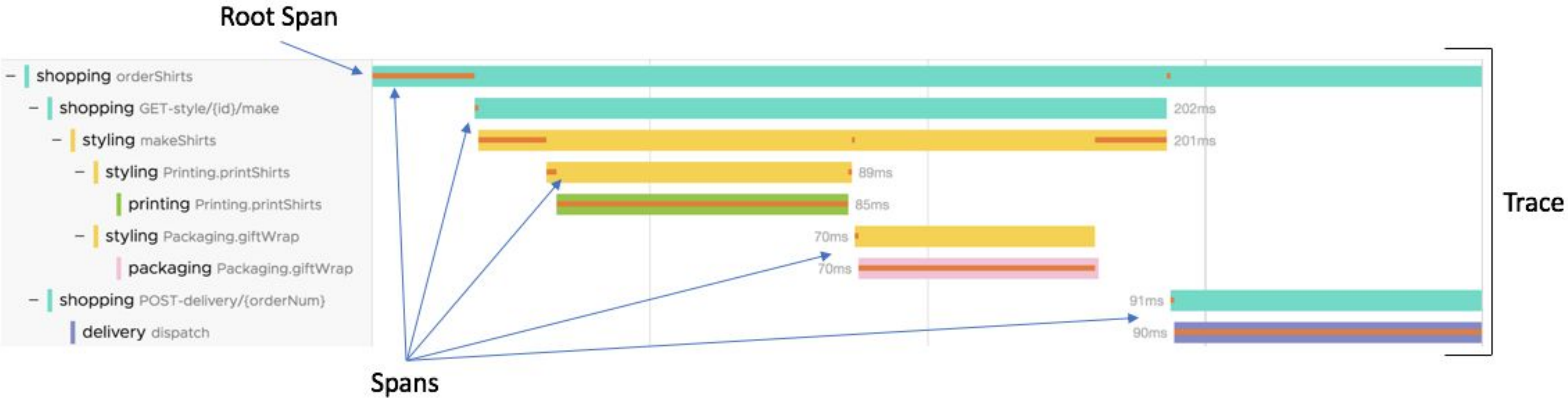
Birds Eye view

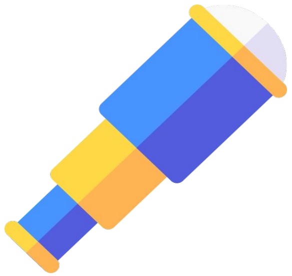


Our focus


Microservices
OTel Auto. Inst.
OTel API
OTel SDK
Shared Infra
Kubernetes
L7 Proxy
aws
3rd party service
OTel Collector
Client Instrumentation
Managed DBs
APIs
Observability Frontends & APIs
Time Series Databases
Trace Databases
Column Stores

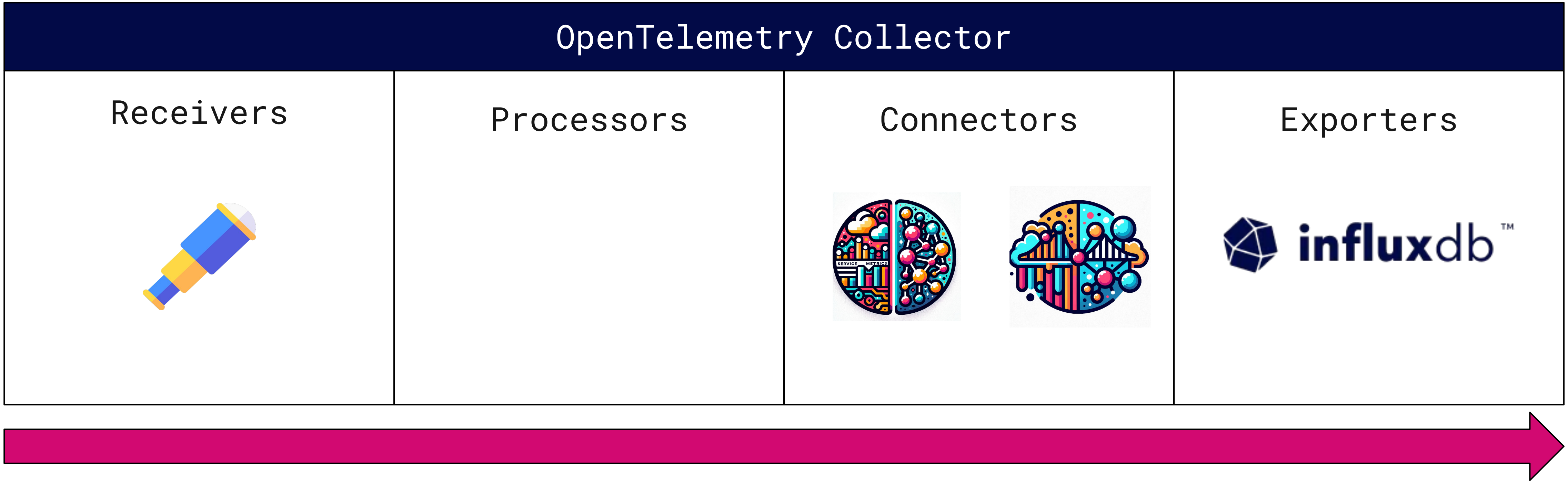
What exactly is a trace?

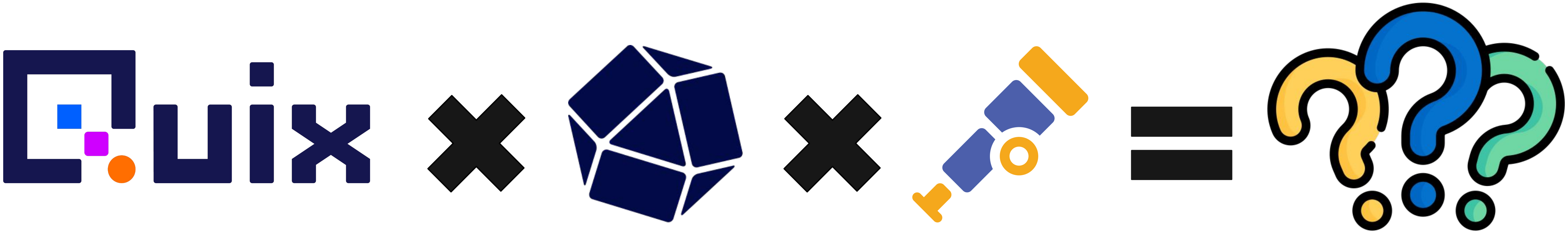

Root Span
– shopping orderShirts
– shopping GET-style/{id}/make
– styling makeShirts
– styling Printing.printShirts
printing Printing.printShirts
– styling Packaging.giftWrap
packaging Packaging.giftWrap
– shopping POST-delivery/{orderNum}
delivery dispatch
202ms
201ms
89ms
85ms
70ms
70ms
91ms
90ms
Trace
Spans



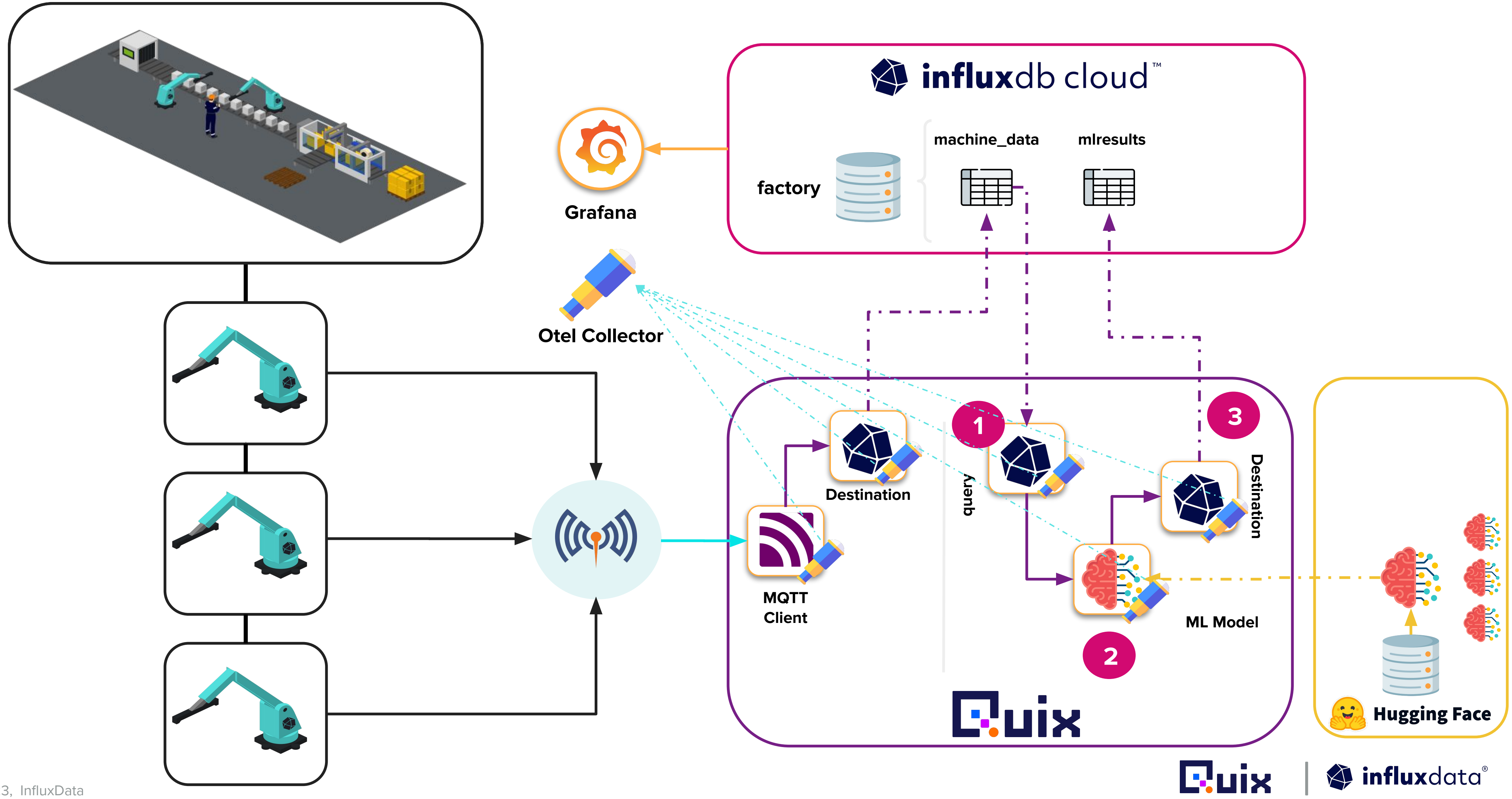
OpenTelemetry Collector

An open source agent that facilitates the collection, processing and export of telemetry data.




Quix

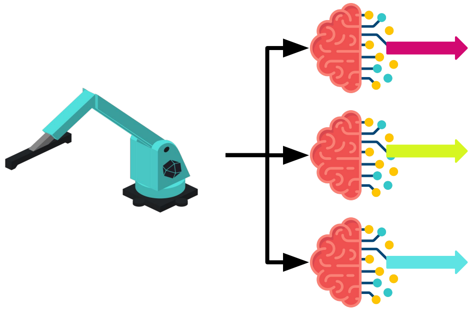
Solution Architecture (Otel)



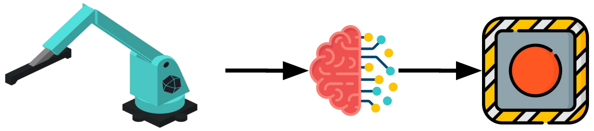
Next Steps

Where could we go next?

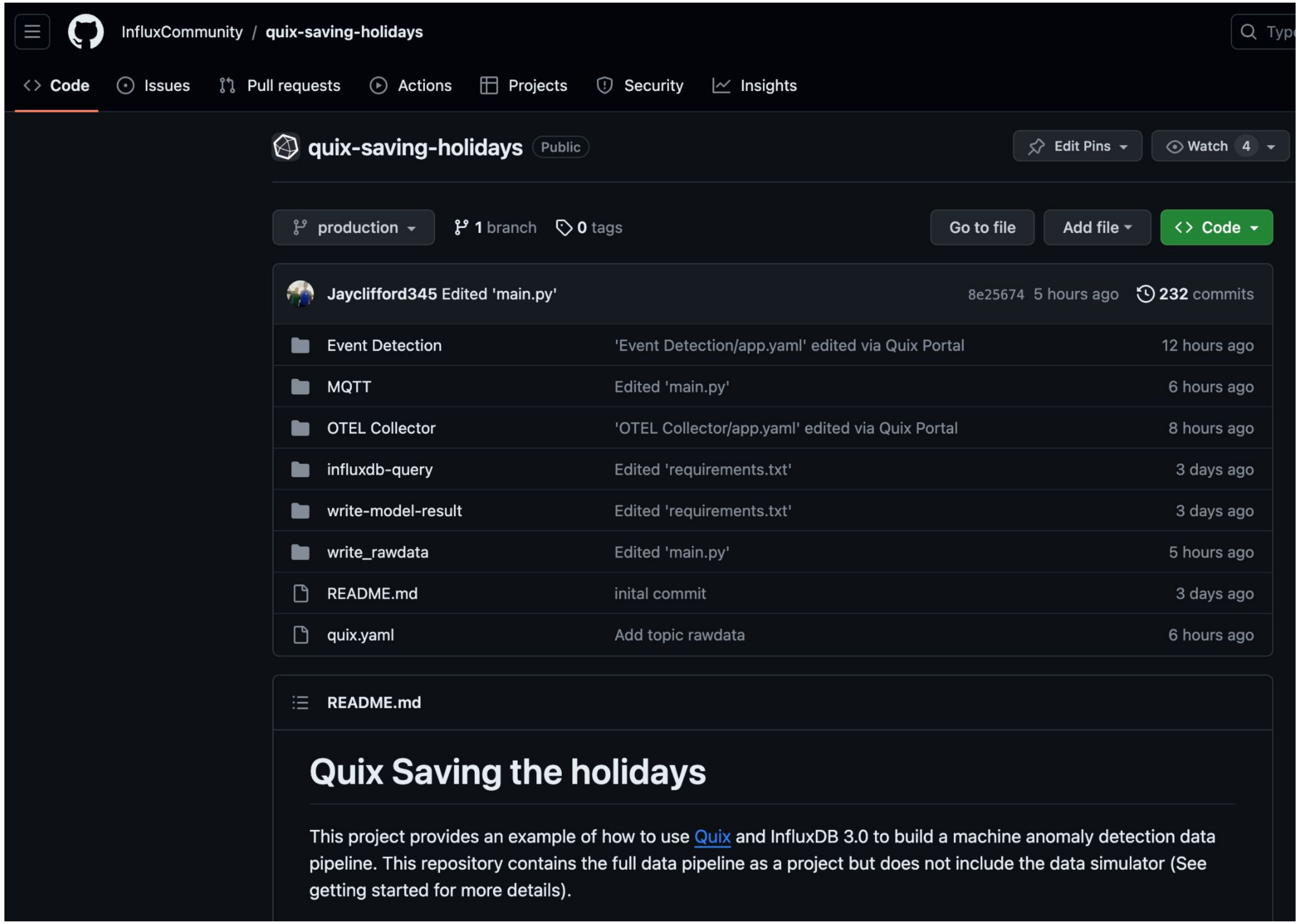
Parallel Model Deployment



Closing the loop (Eventually)



Try the demo yourself



quix-saving-holidays

Jayclifford345 Edited 'main.py' 8e25674 5 hours ago 232 commits

Event Detection	'Event Detection/app.yaml' edited via Quix Portal	12 hours ago
MQTT	Edited 'main.py'	6 hours ago
OTEL Collector	'OTEL Collector/app.yaml' edited via Quix Portal	8 hours ago
influxdb-query	Edited 'requirements.txt'	3 days ago
write-model-result	Edited 'requirements.txt'	3 days ago
write_rawdata	Edited 'main.py'	5 hours ago
README.md	inital commit	3 days ago
quix.yaml	Add topic rawdata	6 hours ago

README.md

Quix Saving the holidays

This project provides an example of how to use Quix and InfluxDB 3.0 to build a machine anomaly detection data pipeline. This repository contains the full data pipeline as a project but does not include the data simulator (See getting started for more details).

https://github.com/InfluxCommunity/quix-saving-holidays

Getting started

Sign up

Influxdata.com ------------------------------------- Get InfluxDB

Via cloud marketplace ------------------- aws Google Cloud Microsoft Azure

Learn

- ☑ Self-service content
- ☑ Documentation
- ☑ InfluxDB University https://influxdbu.com/
- ☑ Community https://influxcommunity.slack.com/

Quix | influxdata®

Getting started

Sign up

quix.io ------------------------------------ Get Quix

Bring your own cloud -------------------- aws Google Cloud Microsoft Azure

Learn

☑ Templates https://quix.io/templates

☑ Documentation https://quix.io/docs

☑ Quix Streams https://github.com/quixio/quix-streams

☑ Community ❤ https://quix.io/slack-invite

THANK YOU

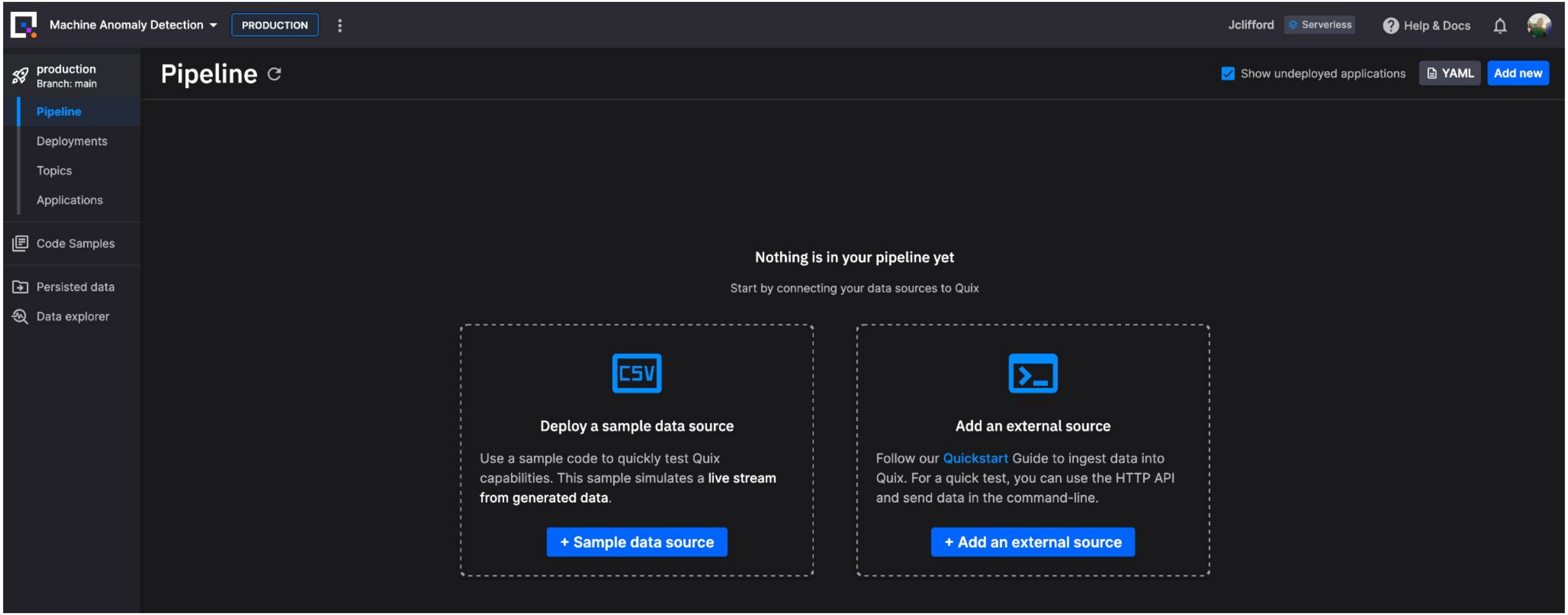
Enjoy the Open Source Data Summit!

influxdata®

www.influxdata.com

Demo Screenshots

New Project



Pipeline

☑ Show undeployed applications

Nothing is in your pipeline yet

Start by connecting your data sources to Quix

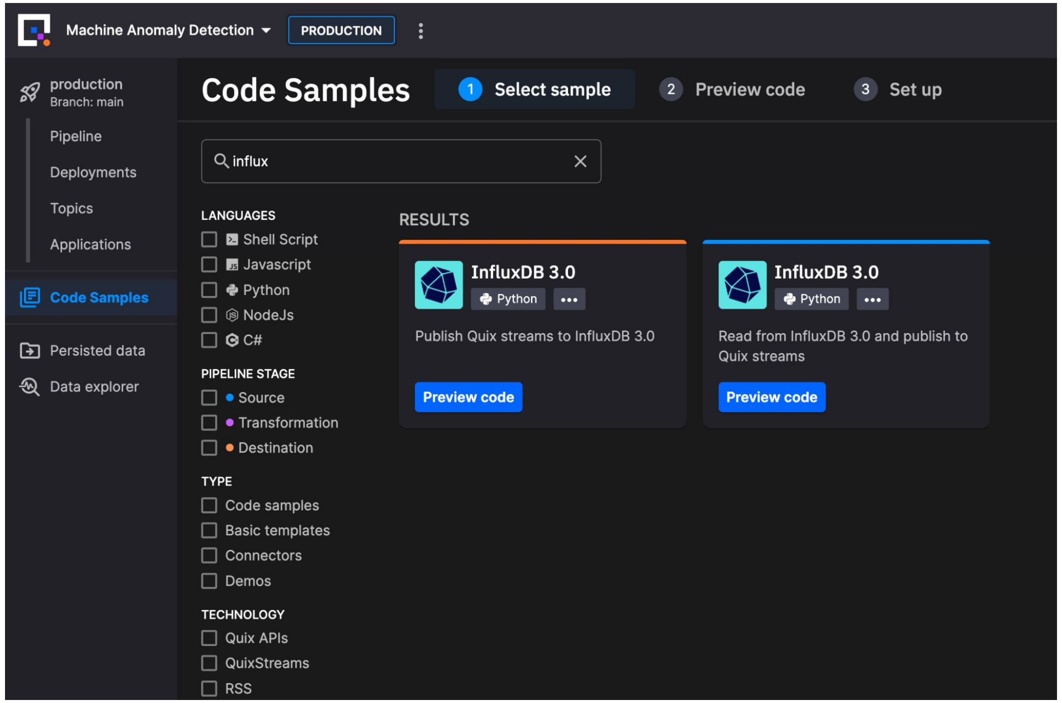
Deploy a sample data source

Use a sample code to quickly test Quix capabilities. This sample simulates a **live stream from generated data**.

Add an external source

Follow our Quickstart Guide to ingest data into Quix. For a quick test, you can use the HTTP API and send data in the command-line.

MQTT ➔ InfluxDB

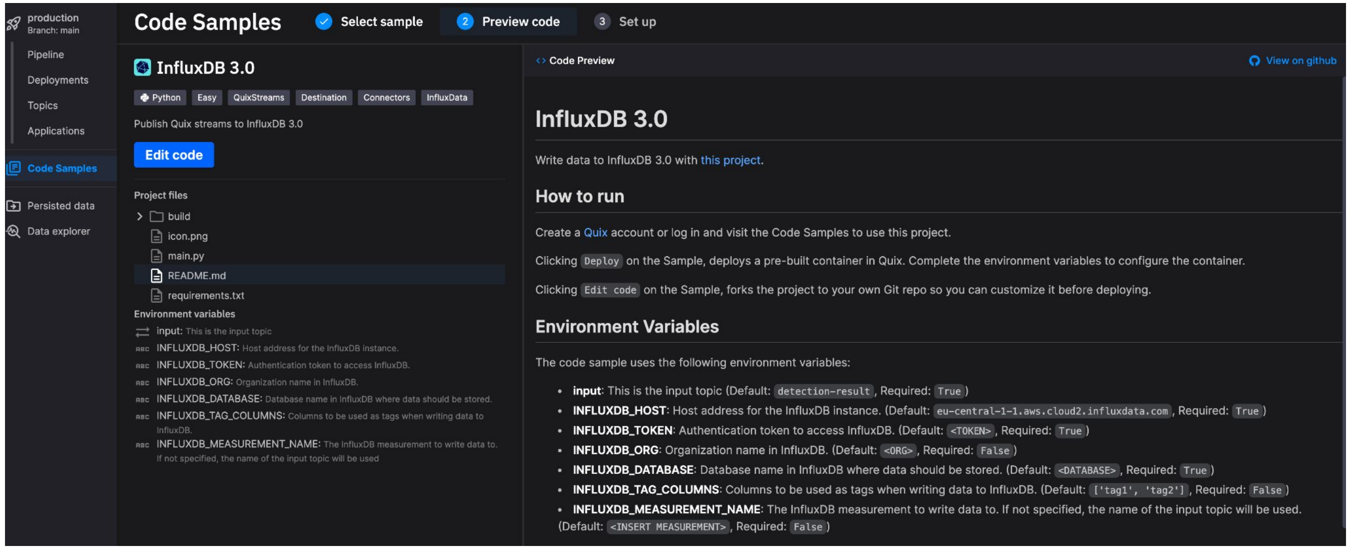


InfluxDB 3.0

Publish Quix streams to InfluxDB 3.0

InfluxDB 3.0

Read from InfluxDB 3.0 and publish to Quix streams



InfluxDB 3.0

Publish Quix streams to InfluxDB 3.0

Project files

- build
- icon.png
- main.py
- README.md
- requirements.txt

Environment variables

- input: This is the input topic
- INFLUXDB_HOST: Host address for the InfluxDB instance.
- INFLUXDB_TOKEN: Authentication token to access InfluxDB.
- INFLUXDB_ORG: Organization name in InfluxDB.
- INFLUXDB_DATABASE: Database name in InfluxDB where data should be stored.
- INFLUXDB_TAG_COLUMNS: Columns to be used as tags when writing data to InfluxDB.
- INFLUXDB_MEASUREMENT_NAME: The InfluxDB measurement to write data to. If not specified, the name of the input topic will be used

InfluxDB 3.0

Write data to InfluxDB 3.0 with this project.

How to run

Create a Quix account or log in and visit the Code Samples to use this project.

Clicking `Deploy` on the Sample, deploys a pre-built container in Quix. Complete the environment variables to configure the container.

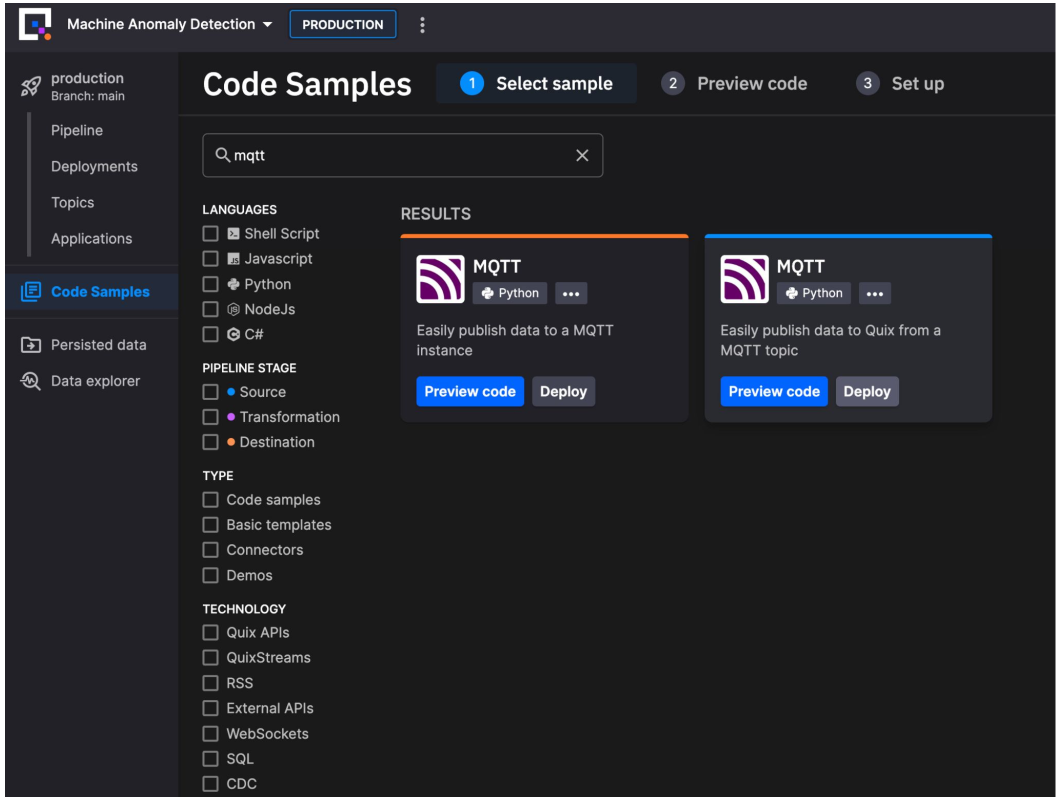
Clicking `Edit code` on the Sample, forks the project to your own Git repo so you can customize it before deploying.

Environment Variables

The code sample uses the following environment variables:

- **input**: This is the input topic (Default: `detection-result`, Required: `True`)
- **INFLUXDB_HOST**: Host address for the InfluxDB instance. (Default: `eu-central-1-1.aws.cloud2.influxdata.com`, Required: `True`)
- **INFLUXDB_TOKEN**: Authentication token to access InfluxDB. (Default: `<TOKEN>`, Required: `True`)
- **INFLUXDB_ORG**: Organization name in InfluxDB. (Default: `<ORG>`, Required: `False`)
- **INFLUXDB_DATABASE**: Database name in InfluxDB where data should be stored. (Default: `<DATABASE>`, Required: `True`)
- **INFLUXDB_TAG_COLUMNS**: Columns to be used as tags when writing data to InfluxDB. (Default: `['tag1', 'tag2']`, Required: `False`)
- **INFLUXDB_MEASUREMENT_NAME**: The InfluxDB measurement to write data to. If not specified, the name of the input topic will be used. (Default: `<INSERT MEASUREMENT>`, Required: `False`)

MQTT ➜ InfluxDB



Machine Anomaly Detection ▾ PRODUCTION ⋮

production
Branch: main

- Pipeline
- Deployments
- Topics
- Applications
- Code Samples
- Persisted data
- Data explorer

Code Samples

1 Select sample　2 Preview code　3 Set up

mqtt ✕

LANGUAGES

□ Shell Script
□ Javascript
□ Python
□ NodeJs
□ C#

PIPELINE STAGE

□ ● Source
□ ● Transformation
□ ● Destination

TYPE

□ Code samples
□ Basic templates
□ Connectors
□ Demos

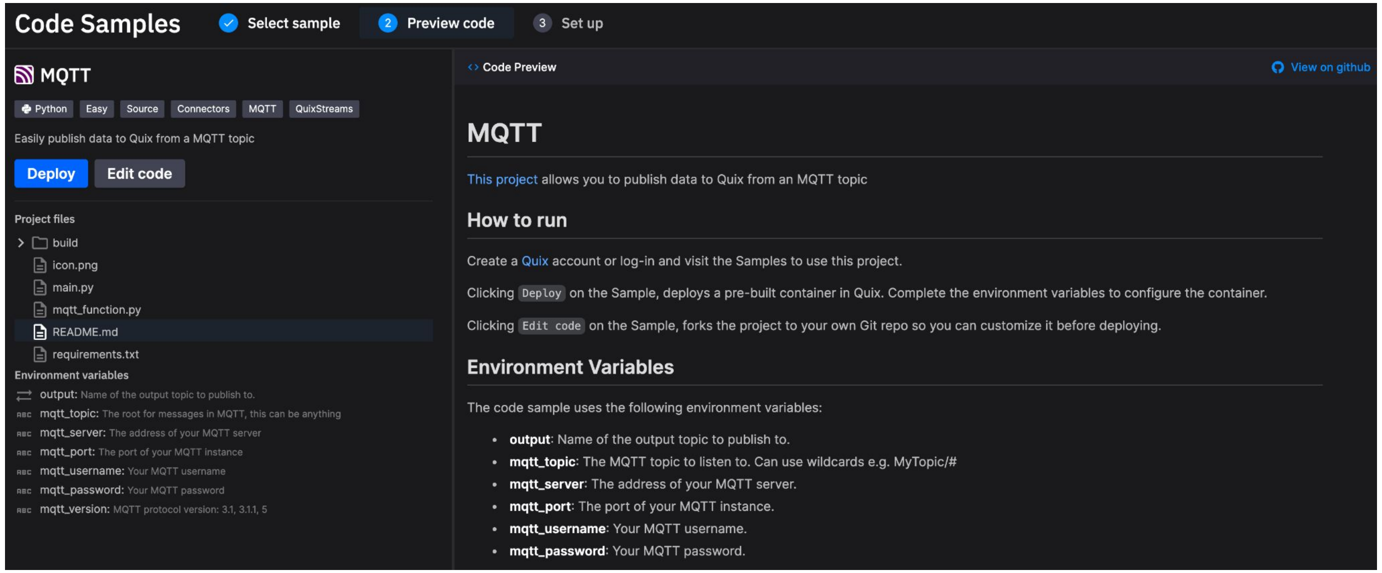
TECHNOLOGY

□ Quix APIs
□ QuixStreams
□ RSS
□ External APIs
□ WebSockets
□ SQL
□ CDC

RESULTS

MQTT
Python •••
Easily publish data to a MQTT instance
Preview code　Deploy

MQTT
Python •••
Easily publish data to Quix from a MQTT topic
Preview code　Deploy



Code Samples

✓ Select sample　2 Preview code　3 Set up

MQTT

Python　Easy　Source　Connectors　MQTT　QuixStreams

Easily publish data to Quix from a MQTT topic

Deploy　Edit code

Project files

- build
- icon.png
- main.py
- mqtt_function.py
- README.md
- requirements.txt

Environment variables

- output: Name of the output topic to publish to.
- mqtt_topic: The root for messages in MQTT, this can be anything
- mqtt_server: The address of your MQTT server
- mqtt_port: The port of your MQTT instance
- mqtt_username: Your MQTT username
- mqtt_password: Your MQTT password
- mqtt_version: MQTT protocol version: 3.1, 3.1.1, 5

<> Code Preview　View on github

MQTT

This project allows you to publish data to Quix from an MQTT topic

How to run

Create a Quix account or log-in and visit the Samples to use this project.

Clicking `Deploy` on the Sample, deploys a pre-built container in Quix. Complete the environment variables to configure the container.

Clicking `Edit code` on the Sample, forks the project to your own Git repo so you can customize it before deploying.

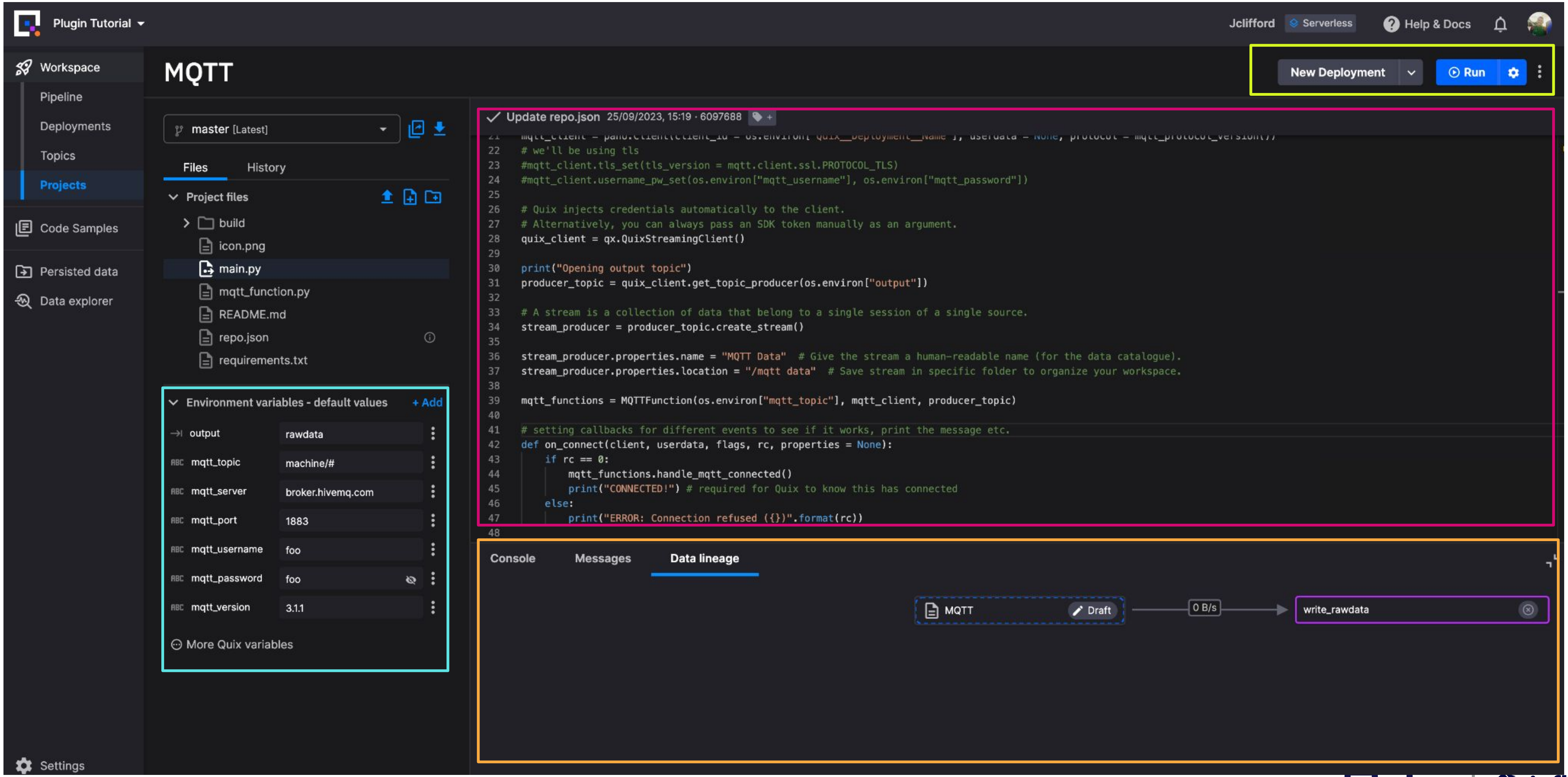
Environment Variables

The code sample uses the following environment variables:

- **output**: Name of the output topic to publish to.
- **mqtt_topic**: The MQTT topic to listen to. Can use wildcards e.g. MyTopic/#
- **mqtt_server**: The address of your MQTT server.
- **mqtt_port**: The port of your MQTT instance.
- **mqtt_username**: Your MQTT username.
- **mqtt_password**: Your MQTT password.

Quix | influxdata®

MQTT ➔ InfluxDB



Environment variables - default values + Add

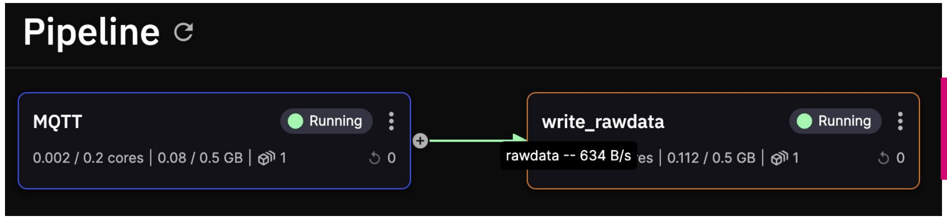
output	rawdata
mqtt_topic	machine/#
mqtt_server	broker.hivemq.com
mqtt_port	1883
mqtt_username	foo
mqtt_password	foo
mqtt_version	3.1.1

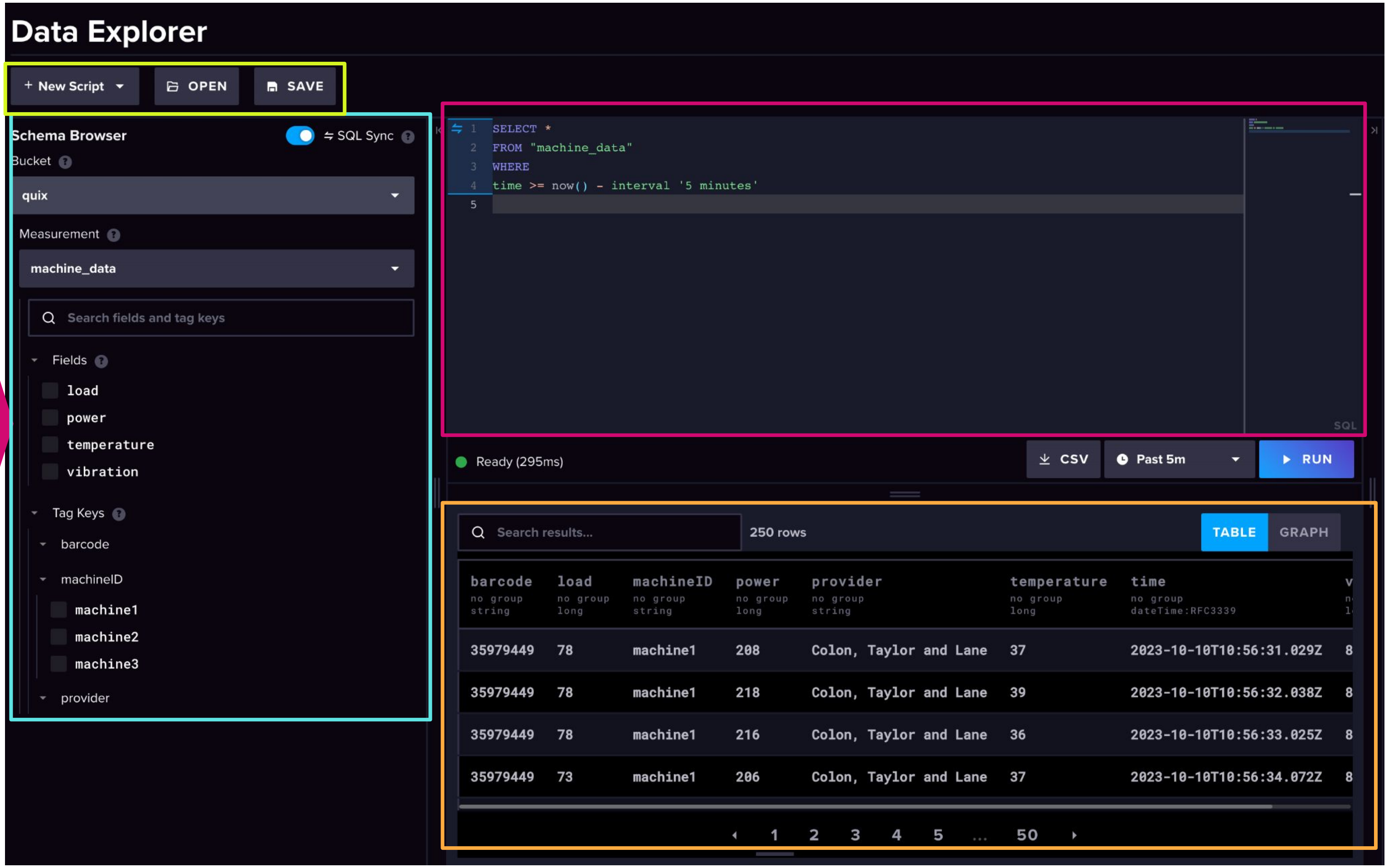
More Quix variables

Update repo.json 25/09/2023, 15:19 · 6097688

```
mqtt_client = paho.Client(client_id = os.environ["Quix__Deployment__Name"], userdata = None, protocol = mqtt_protocol_version())
# we'll be using tls
#mqtt_client.tls_set(tls_version = mqtt.client.ssl.PROTOCOL_TLS)
#mqtt_client.username_pw_set(os.environ["mqtt_username"], os.environ["mqtt_password"])

# Quix injects credentials automatically to the client.
# Alternatively, you can always pass an SDK token manually as an argument.
quix_client = qx.QuixStreamingClient()

print("Opening output topic")
producer_topic = quix_client.get_topic_producer(os.environ["output"])

# A stream is a collection of data that belong to a single session of a single source.
stream_producer = producer_topic.create_stream()

stream_producer.properties.name = "MQTT Data"  # Give the stream a human-readable name (for the data catalogue).
stream_producer.properties.location = "/mqtt data"  # Save stream in specific folder to organize your workspace.

mqtt_functions = MQTTFunction(os.environ["mqtt_topic"], mqtt_client, producer_topic)

# setting callbacks for different events to see if it works, print the message etc.
def on_connect(client, userdata, flags, rc, properties = None):
    if rc == 0:
        mqtt_functions.handle_mqtt_connected()
        print("CONNECTED!") # required for Quix to know this has connected
    else:
        print("ERROR: Connection refused ({})".format(rc))

```

MQTT → InfluxDB





Bucket
quix

Measurement
machine_data

```
SELECT *
FROM "machine_data"
WHERE
time >= now() - interval '5 minutes'
```

Ready (295ms)

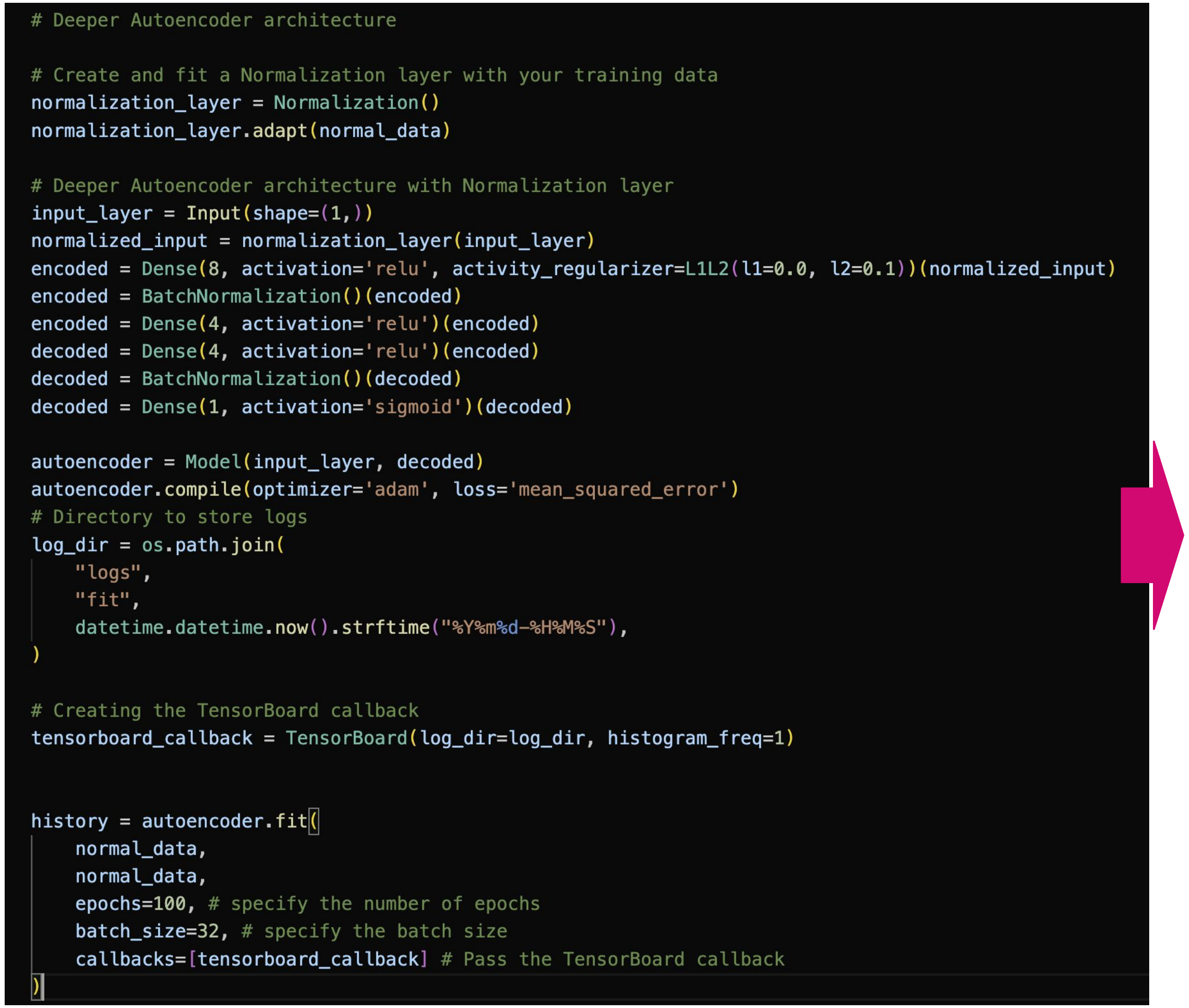
250 rows

barcode	load	machineID	power	provider	temperature	time	v
no group	no group	no group	no group	no group	no group	no group	n
string	long	string	long	string	long	dateTime:RFC3339	l
35979449	78	machine1	208	Colon, Taylor and Lane	37	2023-10-10T10:56:31.029Z	8
35979449	78	machine1	218	Colon, Taylor and Lane	39	2023-10-10T10:56:32.038Z	8
35979449	78	machine1	216	Colon, Taylor and Lane	36	2023-10-10T10:56:33.025Z	8
35979449	73	machine1	206	Colon, Taylor and Lane	37	2023-10-10T10:56:34.072Z	8

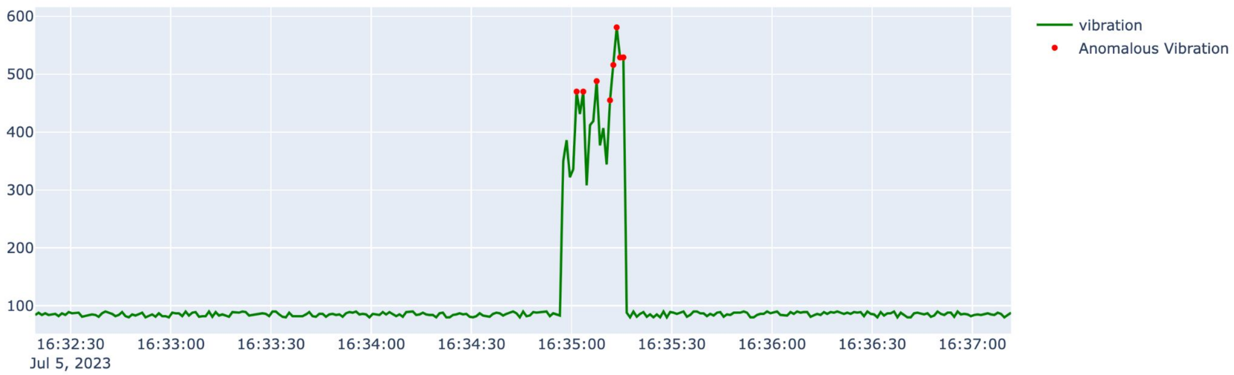
‹ 1 2 3 4 5 … 50 ›

Quix | influxdata®

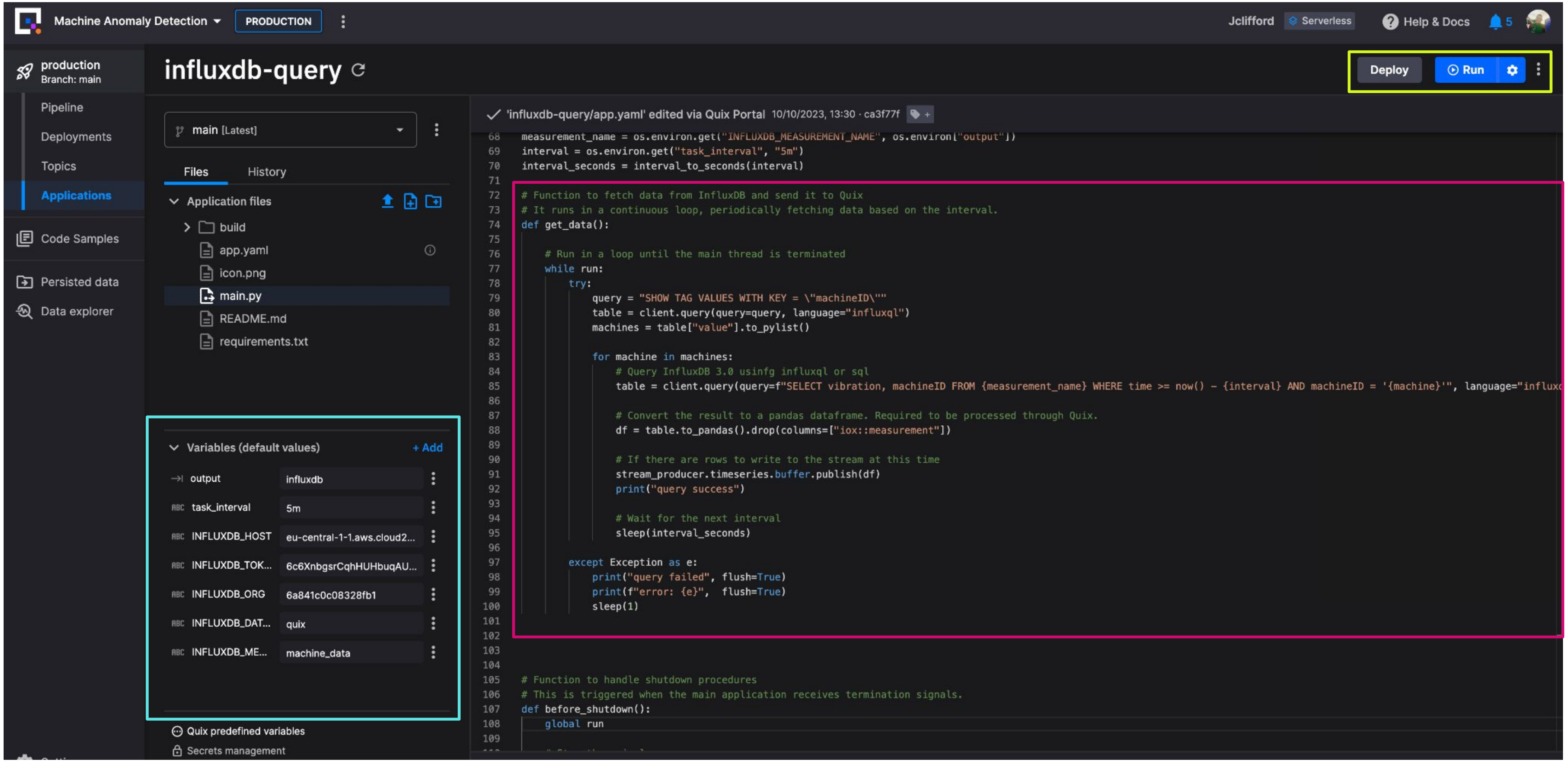
Training



```
# Deeper Autoencoder architecture

# Create and fit a Normalization layer with your training data
normalization_layer = Normalization()
normalization_layer.adapt(normal_data)

# Deeper Autoencoder architecture with Normalization layer
input_layer = Input(shape=(1,))
normalized_input = normalization_layer(input_layer)
encoded = Dense(8, activation='relu', activity_regularizer=L1L2(l1=0.0, l2=0.1))(normalized_input)
encoded = BatchNormalization()(encoded)
encoded = Dense(4, activation='relu')(encoded)
decoded = Dense(4, activation='relu')(encoded)
decoded = BatchNormalization()(decoded)
decoded = Dense(1, activation='sigmoid')(decoded)

autoencoder = Model(input_layer, decoded)
autoencoder.compile(optimizer='adam', loss='mean_squared_error')
# Directory to store logs
log_dir = os.path.join(
    "logs",
    "fit",
    datetime.datetime.now().strftime("%Y%m%d-%H%M%S"),
)

# Creating the TensorBoard callback
tensorboard_callback = TensorBoard(log_dir=log_dir, histogram_freq=1)


history = autoencoder.fit(
    normal_data,
    normal_data,
    epochs=100, # specify the number of epochs
    batch_size=32, # specify the batch size
    callbacks=[tensorboard_callback] # Pass the TensorBoard callback
)
```



InfluxDB → Quix



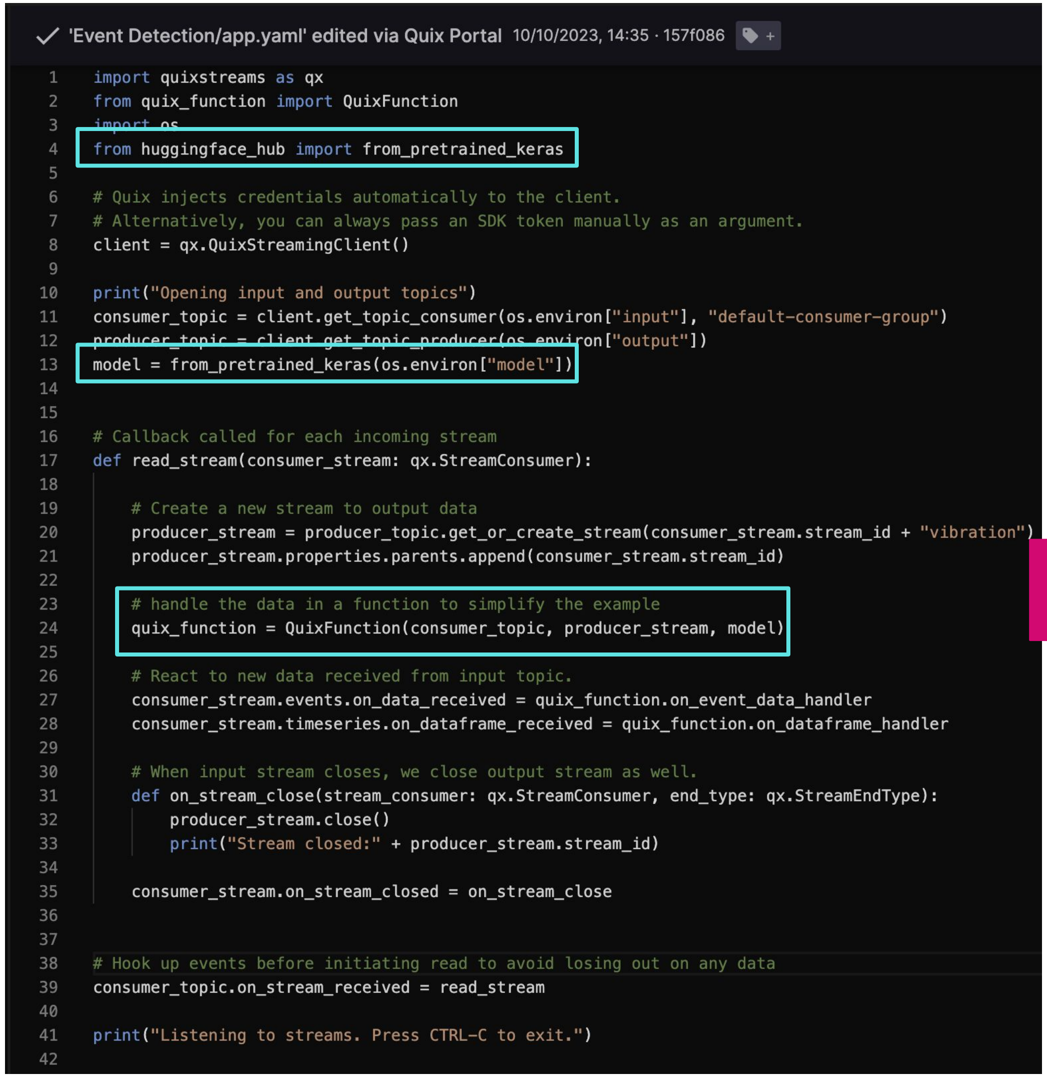
'influxdb-query/app.yaml' edited via Quix Portal 10/10/2023, 13:30 · ca3f77f

Variables (default values) + Add

output	influxdb
task_interval	5m
INFLUXDB_HOST	eu-central-1-1.aws.cloud2...
INFLUXDB_TOK...	6c6XnbgsrCqhHUHbuqAU...
INFLUXDB_ORG	6a841c0c08328fb1
INFLUXDB_DAT...	quix
INFLUXDB_ME...	machine_data

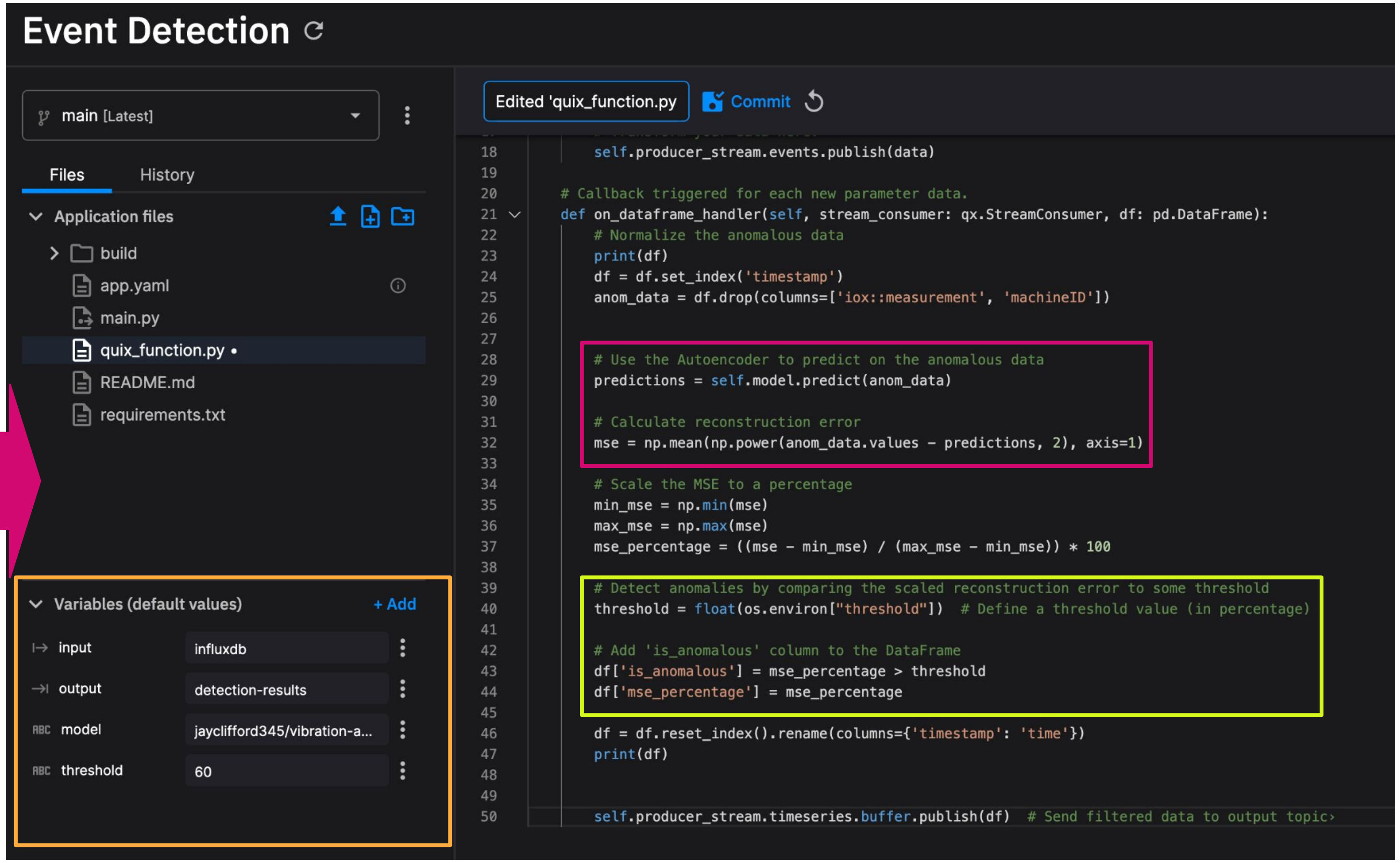
```python
measurement_name = os.environ.get("INFLUXDB_MEASUREMENT_NAME", os.environ["output"])
interval = os.environ.get("task_interval", "5m")
interval_seconds = interval_to_seconds(interval)

# Function to fetch data from InfluxDB and send it to Quix
# It runs in a continuous loop, periodically fetching data based on the interval.
def get_data():

    # Run in a loop until the main thread is terminated
    while run:
        try:
            query = "SHOW TAG VALUES WITH KEY = \"machineID\""
            table = client.query(query=query, language="influxql")
            machines = table["value"].to_pylist()

            for machine in machines:
                # Query InfluxDB 3.0 usinfg influxql or sql
                table = client.query(query=f"SELECT vibration, machineID FROM {measurement_name} WHERE time >= now() - {interval} AND machineID = '{machine}'", language="influx

                # Convert the result to a pandas dataframe. Required to be processed through Quix.
                df = table.to_pandas().drop(columns=["iox::measurement"])

                # If there are rows to write to the stream at this time
                stream_producer.timeseries.buffer.publish(df)
                print("query success")

                # Wait for the next interval
                sleep(interval_seconds)

        except Exception as e:
            print("query failed", flush=True)
            print(f"error: {e}",  flush=True)
            sleep(1)



# Function to handle shutdown procedures
# This is triggered when the main application receives termination signals.
def before_shutdown():
    global run

```

Quix ➔ Model ➔ Quix



'Event Detection/app.yaml' edited via Quix Portal 10/10/2023, 14:35 · 157f086

```python
import quixstreams as qx
from quix_function import QuixFunction
import os
from huggingface_hub import from_pretrained_keras

# Quix injects credentials automatically to the client.
# Alternatively, you can always pass an SDK token manually as an argument.
client = qx.QuixStreamingClient()

print("Opening input and output topics")
consumer_topic = client.get_topic_consumer(os.environ["input"], "default-consumer-group")
producer_topic = client.get_topic_producer(os.environ["output"])
model = from_pretrained_keras(os.environ["model"])


# Callback called for each incoming stream
def read_stream(consumer_stream: qx.StreamConsumer):

    # Create a new stream to output data
    producer_stream = producer_topic.get_or_create_stream(consumer_stream.stream_id + "vibration")
    producer_stream.properties.parents.append(consumer_stream.stream_id)

    # handle the data in a function to simplify the example
    quix_function = QuixFunction(consumer_topic, producer_stream, model)

    # React to new data received from input topic.
    consumer_stream.events.on_data_received = quix_function.on_event_data_handler
    consumer_stream.timeseries.on_dataframe_received = quix_function.on_dataframe_handler

    # When input stream closes, we close output stream as well.
    def on_stream_close(stream_consumer: qx.StreamConsumer, end_type: qx.StreamEndType):
        producer_stream.close()
        print("Stream closed:" + producer_stream.stream_id)

    consumer_stream.on_stream_closed = on_stream_close


# Hook up events before initiating read to avoid losing out on any data
consumer_topic.on_stream_received = read_stream

print("Listening to streams. Press CTRL-C to exit.")
```

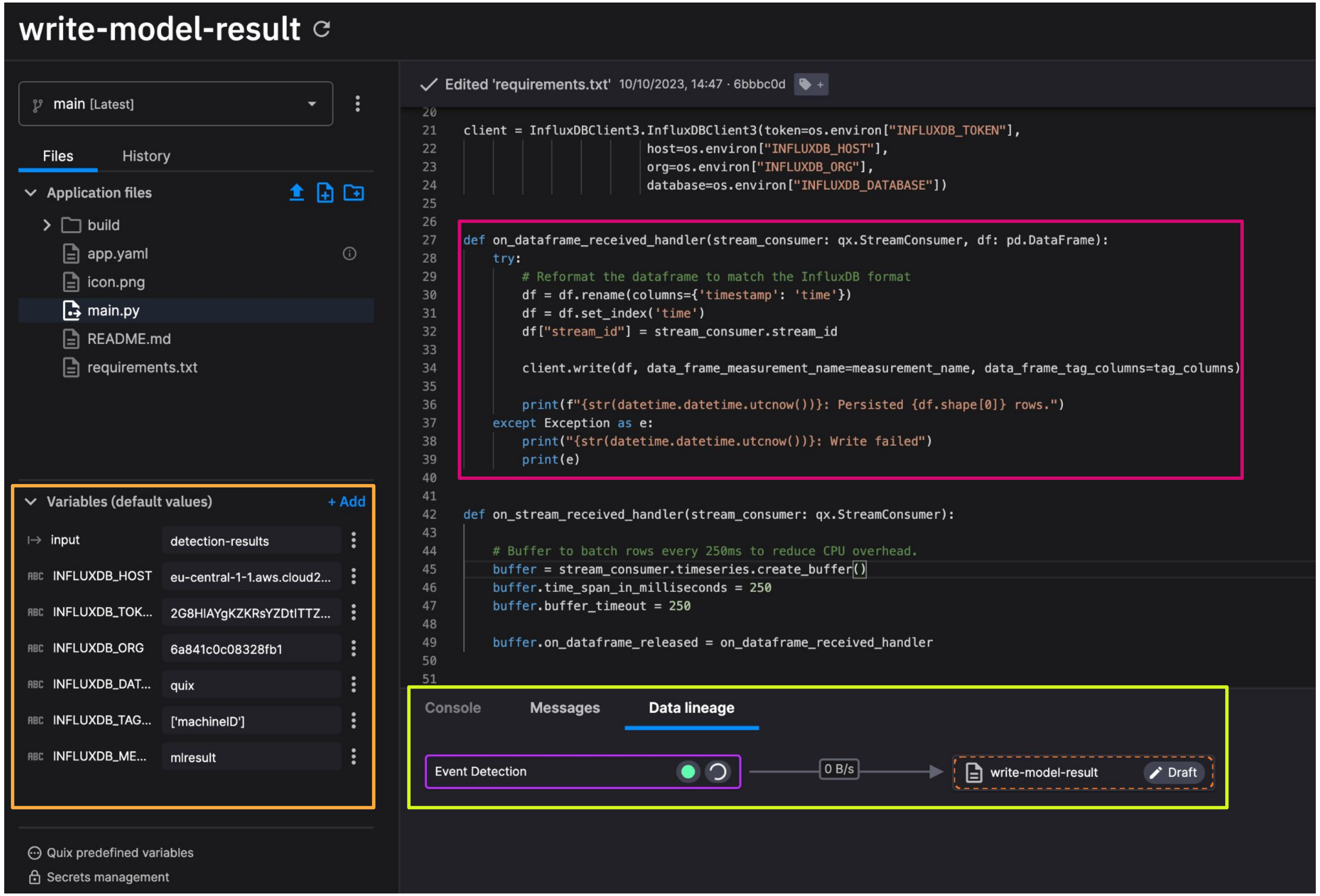


Variables (default values) + Add

input	influxdb
output	detection-results
model	jayclifford345/vibration-a...
threshold	60

```python
        self.producer_stream.events.publish(data)

    # Callback triggered for each new parameter data.
    def on_dataframe_handler(self, stream_consumer: qx.StreamConsumer, df: pd.DataFrame):
        # Normalize the anomalous data
        print(df)
        df = df.set_index('timestamp')
        anom_data = df.drop(columns=['iox::measurement', 'machineID'])


        # Use the Autoencoder to predict on the anomalous data
        predictions = self.model.predict(anom_data)

        # Calculate reconstruction error
        mse = np.mean(np.power(anom_data.values - predictions, 2), axis=1)

        # Scale the MSE to a percentage
        min_mse = np.min(mse)
        max_mse = np.max(mse)
        mse_percentage = ((mse - min_mse) / (max_mse - min_mse)) * 100

        # Detect anomalies by comparing the scaled reconstruction error to some threshold
        threshold = float(os.environ["threshold"])  # Define a threshold value (in percentage)

        # Add 'is_anomalous' column to the DataFrame
        df['is_anomalous'] = mse_percentage > threshold
        df['mse_percentage'] = mse_percentage

        df = df.reset_index().rename(columns={'timestamp': 'time'})
        print(df)


        self.producer_stream.timeseries.buffer.publish(df)  # Send filtered data to output topic›
```

Quix ➔ InfluxDB



write-model-result

main [Latest]

Files | History

Application files

- build
- app.yaml
- icon.png
- main.py
- README.md
- requirements.txt

Variables (default values) + Add

input	detection-results
INFLUXDB_HOST	eu-central-1-1.aws.cloud2...
INFLUXDB_TOK...	2G8HIAYgKZKRsYZDtITTZ...
INFLUXDB_ORG	6a841c0c08328fb1
INFLUXDB_DAT...	quix
INFLUXDB_TAG...	['machineID']
INFLUXDB_ME...	mlresult

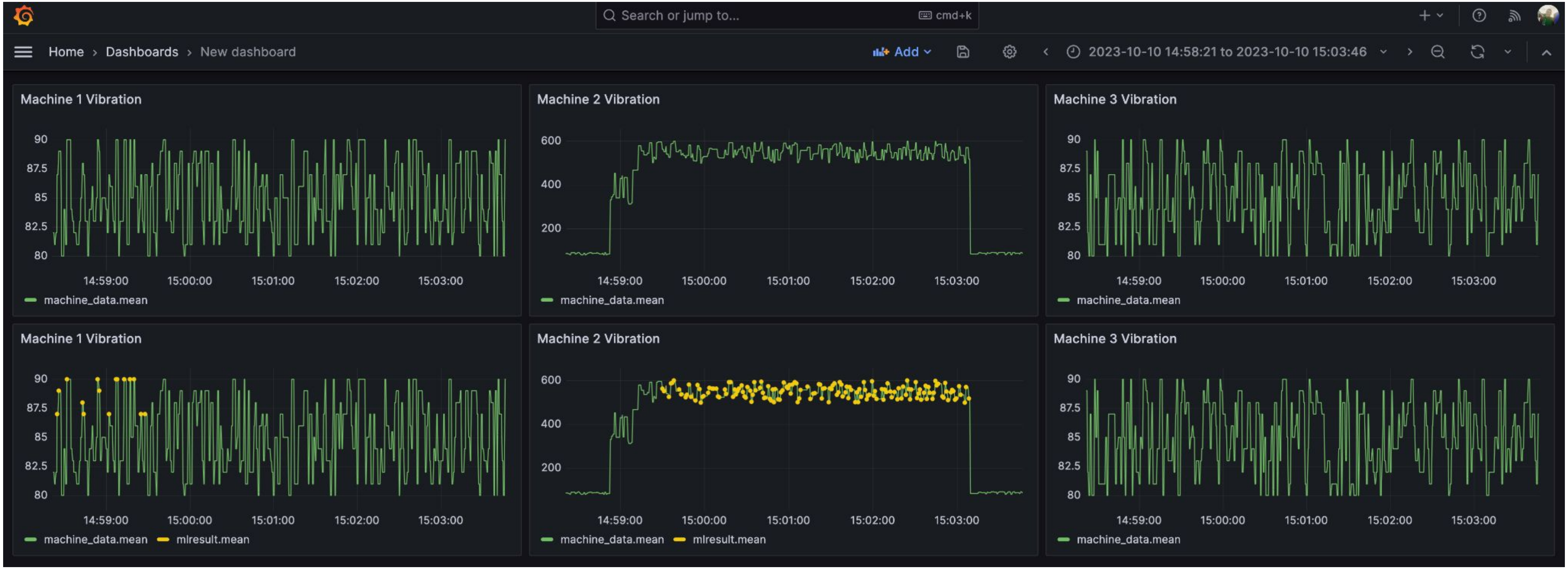
Quix predefined variables

Secrets management

Edited 'requirements.txt' 10/10/2023, 14:47 · 6bbbc0d

```python
client = InfluxDBClient3.InfluxDBClient3(token=os.environ["INFLUXDB_TOKEN"],
                         host=os.environ["INFLUXDB_HOST"],
                         org=os.environ["INFLUXDB_ORG"],
                         database=os.environ["INFLUXDB_DATABASE"])


def on_dataframe_received_handler(stream_consumer: qx.StreamConsumer, df: pd.DataFrame):
    try:
        # Reformat the dataframe to match the InfluxDB format
        df = df.rename(columns={'timestamp': 'time'})
        df = df.set_index('time')
        df["stream_id"] = stream_consumer.stream_id

        client.write(df, data_frame_measurement_name=measurement_name, data_frame_tag_columns=tag_columns)

        print(f"{str(datetime.datetime.utcnow())}: Persisted {df.shape[0]} rows.")
    except Exception as e:
        print("{str(datetime.datetime.utcnow())}: Write failed")
        print(e)


def on_stream_received_handler(stream_consumer: qx.StreamConsumer):

    # Buffer to batch rows every 250ms to reduce CPU overhead.
    buffer = stream_consumer.timeseries.create_buffer()
    buffer.time_span_in_milliseconds = 250
    buffer.buffer_timeout = 250

    buffer.on_dataframe_released = on_dataframe_received_handler

```

InfluxDB ➜ Grafana


Search or jump to...
cmd+k
Home › Dashboards › New dashboard
Add
2023-10-10 14:58:21 to 2023-10-10 15:03:46
Machine 1 Vibration
90
87.5
85
82.5
80
14:59:00
15:00:00
15:01:00
15:02:00
15:03:00
machine_data.mean
Machine 2 Vibration
600
400
200
machine_data.mean
Machine 3 Vibration
machine_data.mean
Machine 1 Vibration
machine_data.mean
mlresult.mean
Machine 2 Vibration
machine_data.mean
mlresult.mean
Machine 3 Vibration
machine_data.mean