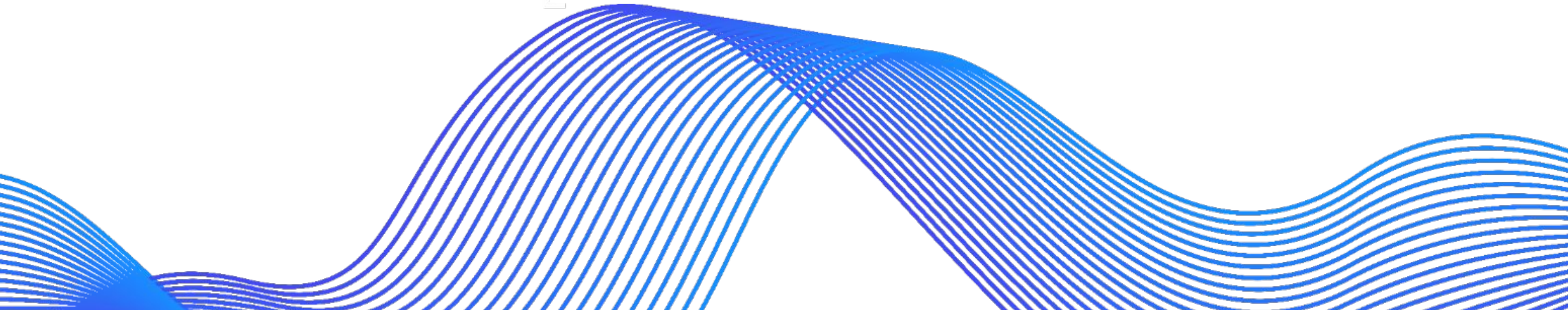
Stream Native

Deep Dive into Building Streaming Applications with Apache Pulsar



Tim Spann
Developer Advocate

- FLiP(N) Stack = Flink, Pulsar and NiFi Stack
- Streaming Systems/ Data Architect
- Experience:
  - 15+ years of experience with batch and streaming technologies including Pulsar, Flink, Spark, NiFi, Spring, Java, Big Data, Cloud, MXNet, Hadoop, Datalakes, IoT and more.

CLOUDERA Pivotal BARNES & NOBLE

FLIP STACK EY HORTONWORKS airis.DATA Hewlett Packard Enterprise

StreamNative

FLiP Stack Weekly

https://bit.ly/32dAJft

This week in Apache Flink, Apache Pulsar, Apache NiFi, Apache Spark and open source friends.

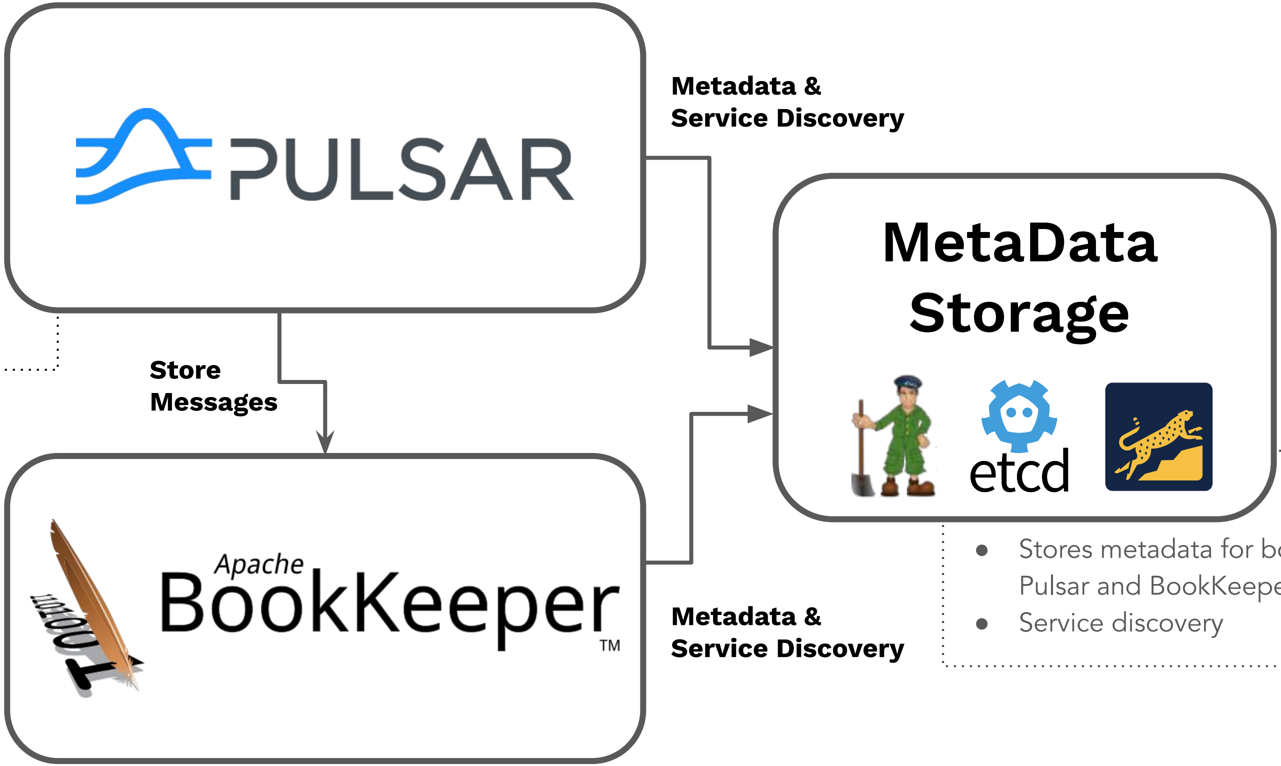
StreamNative

PULSAR

Apache Pulsar is a Cloud-Native Messaging and Event-Streaming Platform.

Key **Pulsar** Concepts: Architecture

- "Brokers"
- Handles message routing and connections
- Stateless, but with caches
- Automatic load-balancing
- Topics are composed of multiple segments



Metadata & Service Discovery

Store Messages

MetaData Storage

Metadata & Service Discovery

- Stores metadata for both Pulsar and BookKeeper
- Service discovery

- "Bookies"
- Stores messages and cursors
- Messages are grouped in segments/ledgers
- A group of bookies form an "ensemble" to store a ledger

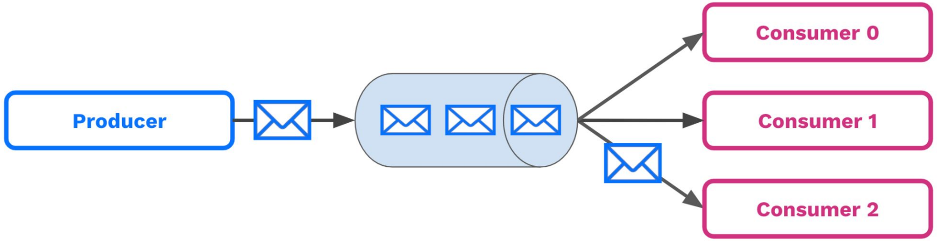
StreamNative

Messages - the basic unit of Pulsar

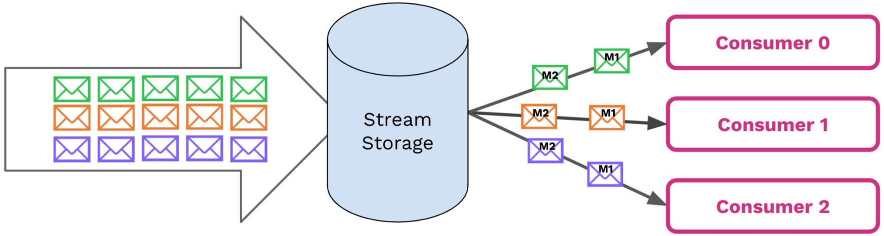
Component	Description
Value / data payload	The data carried by the message. All Pulsar messages contain raw bytes, although message data can also conform to data schemas.
Key	Messages are optionally tagged with keys, used in partitioning and also is useful for things like topic compaction.
Properties	An optional key/value map of user-defined properties.
Producer name	The name of the producer who produces the message. If you do not specify a producer name, the default name is used. Message De-Duplication.
Sequence ID	Each Pulsar message belongs to an ordered sequence on its topic. The sequence ID of the message is its order in that sequence. Message De-Duplication.

Key **Pulsar** Concepts: Messaging vs Streaming

Message Queueing - Queueing systems are ideal for work queues that do not require tasks to be performed in a particular order.

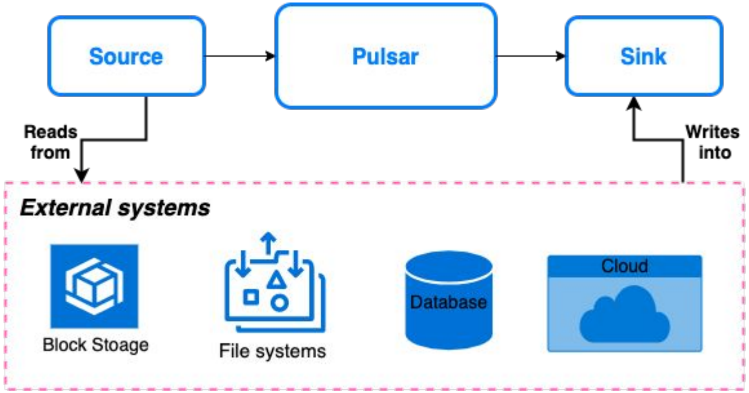


Streaming - Streaming works best in situations where the order of messages is important.



StreamNative

Connectivity

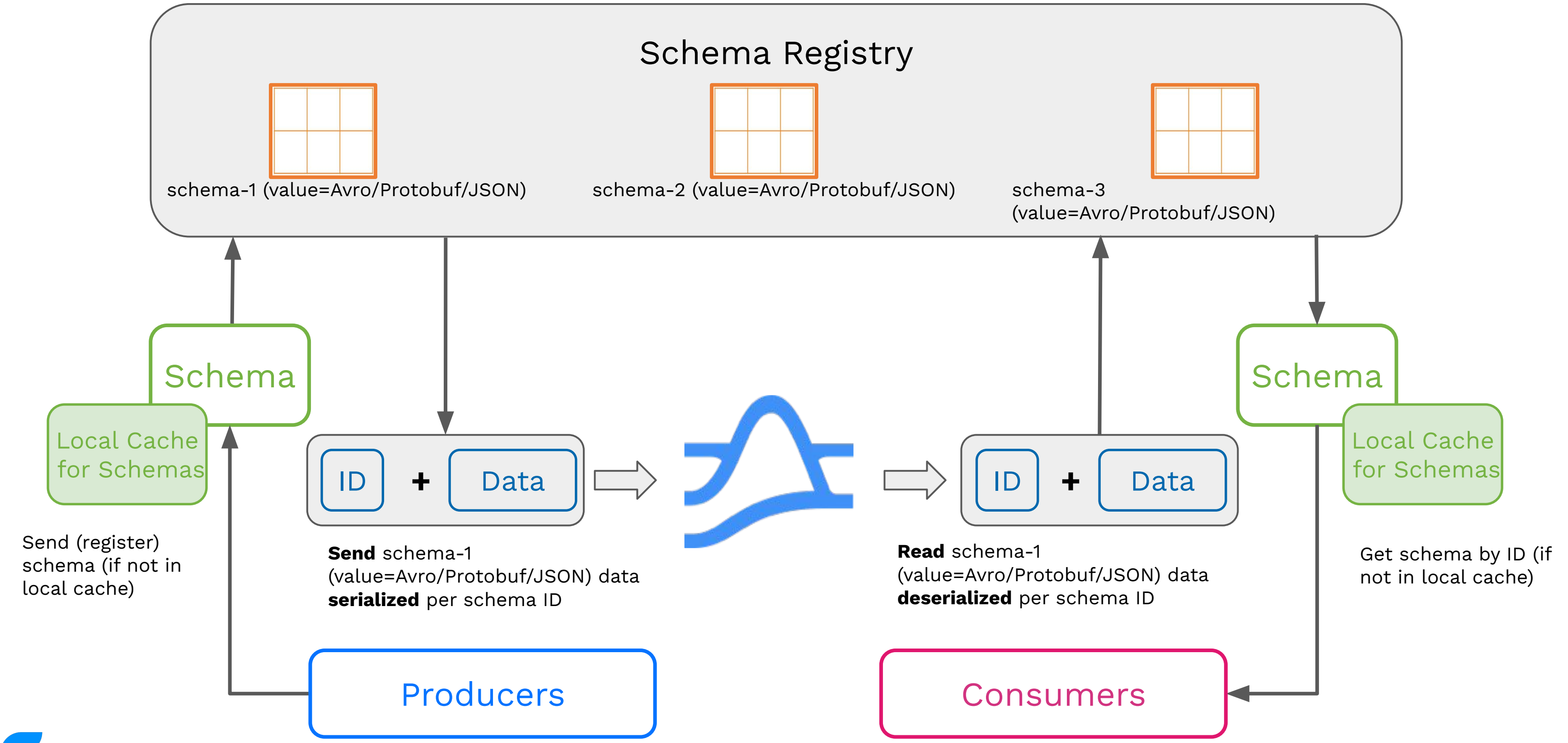


hub.streamnative.io

StreamNative

- **Functions** - Lightweight Stream Processing (Java, Python, Go)
- **Connectors** - Sources & Sinks (Cassandra, Kafka, …)
- **Protocol Handlers** - AoP (AMQP), KoP (Kafka), MoP (MQTT)
- **Processing Engines** - Flink, Spark, Presto/Trino via Pulsar SQL
- **Data Offloaders** - Tiered Storage - (S3)

Schema Registry



Schema Registry
schema-1 (value=Avro/Protobuf/JSON)
schema-2 (value=Avro/Protobuf/JSON)
schema-3 (value=Avro/Protobuf/JSON)
Schema
Local Cache for Schemas
ID
+
Data
ID
+
Data
Schema
Local Cache for Schemas
Producers
Consumers

Send (register) schema (if not in local cache)

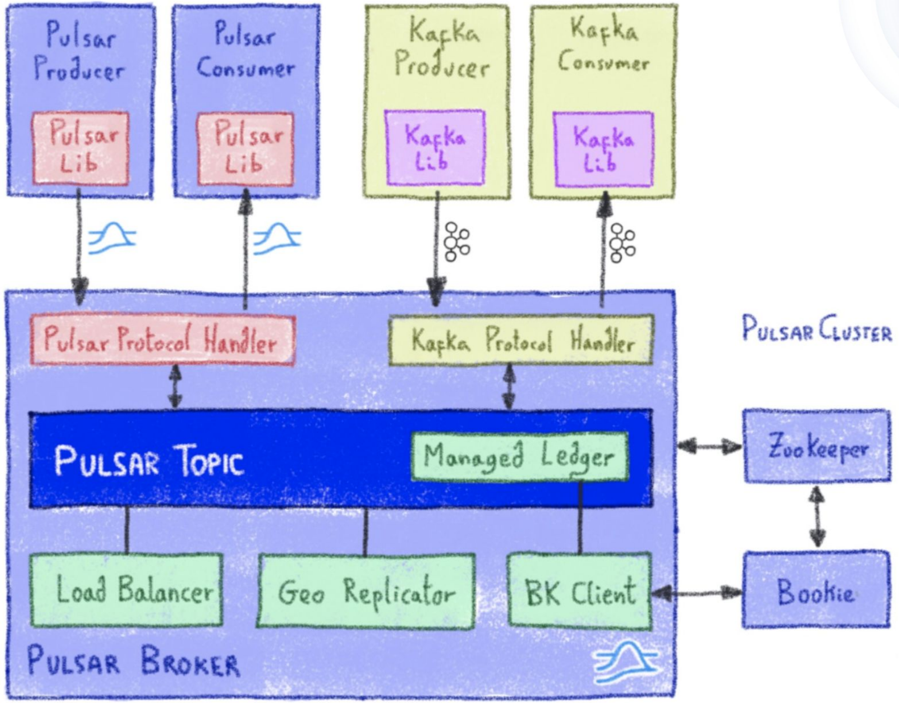
Send schema-1 (value=Avro/Protobuf/JSON) data **serialized** per schema ID

Read schema-1 (value=Avro/Protobuf/JSON) data **deserialized** per schema ID

Get schema by ID (if not in local cache)

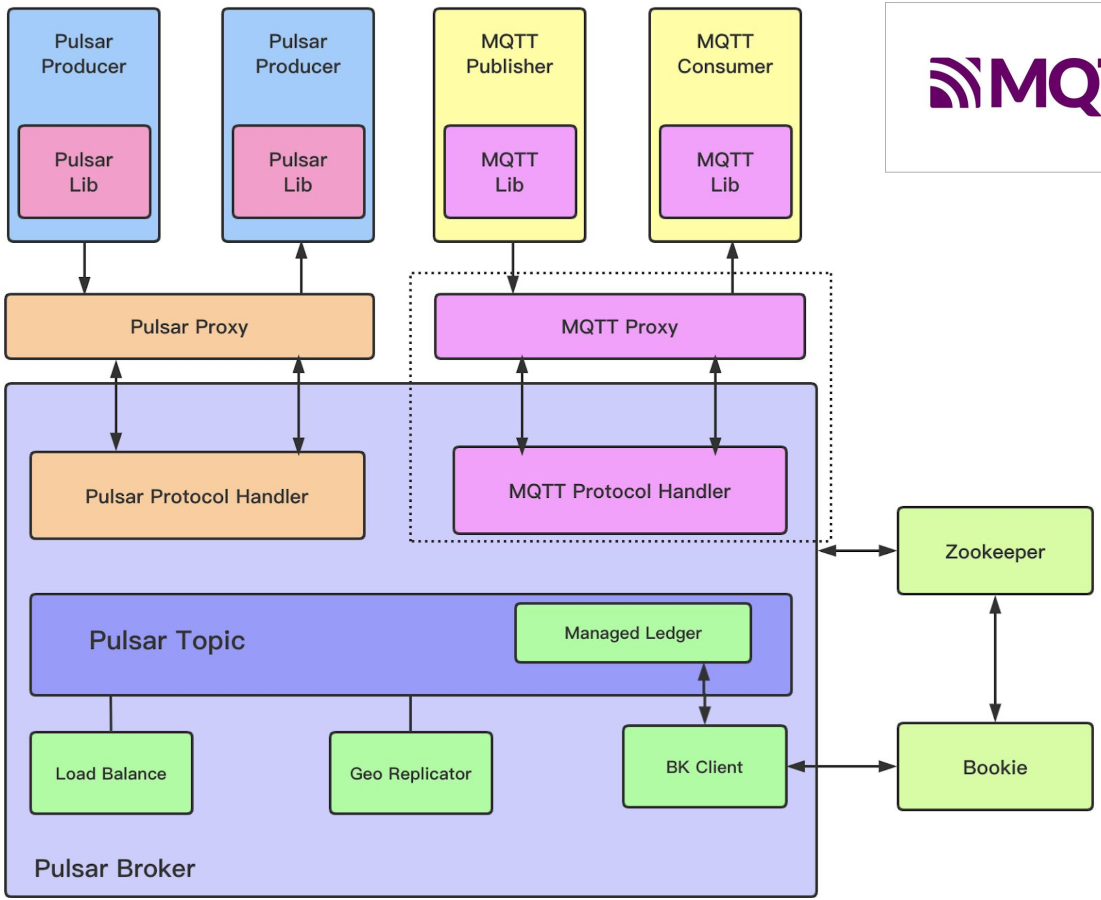
StreamNative

Kafka On Pulsar (KoP)



StreamNative

MQTT On Pulsar (MoP)

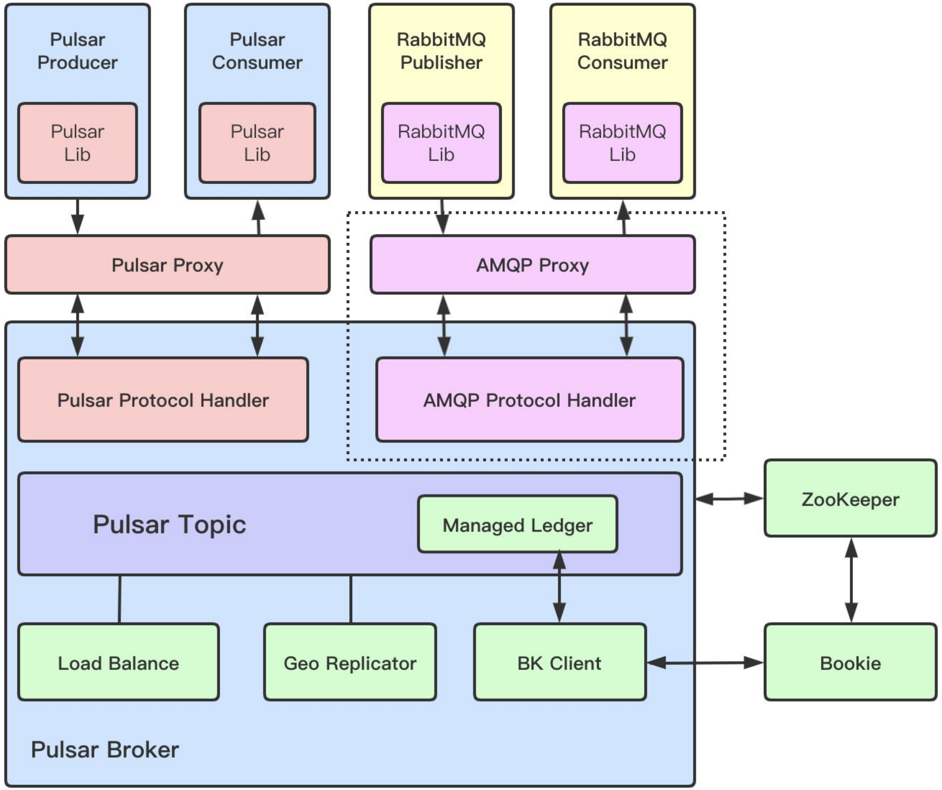


MQTT

StreamNative

AMQP On Pulsar

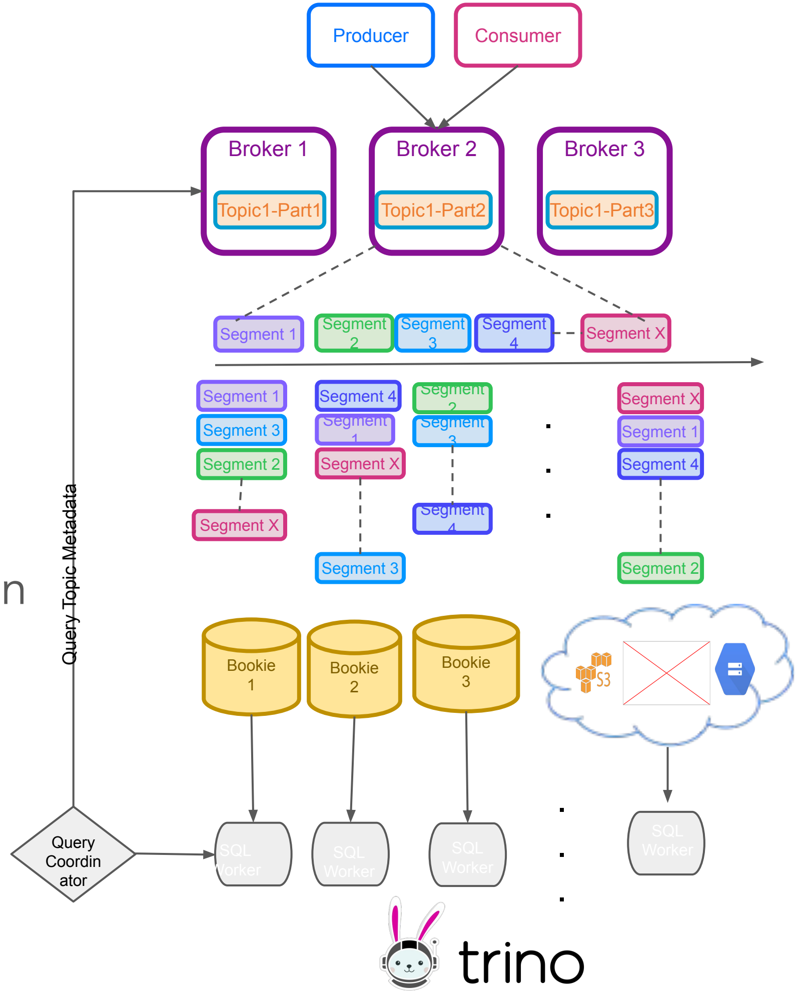
(AoP)



StreamNative

Pulsar SQL

Presto/Trino workers can read segments directly from bookies (or offloaded storage) in parallel.



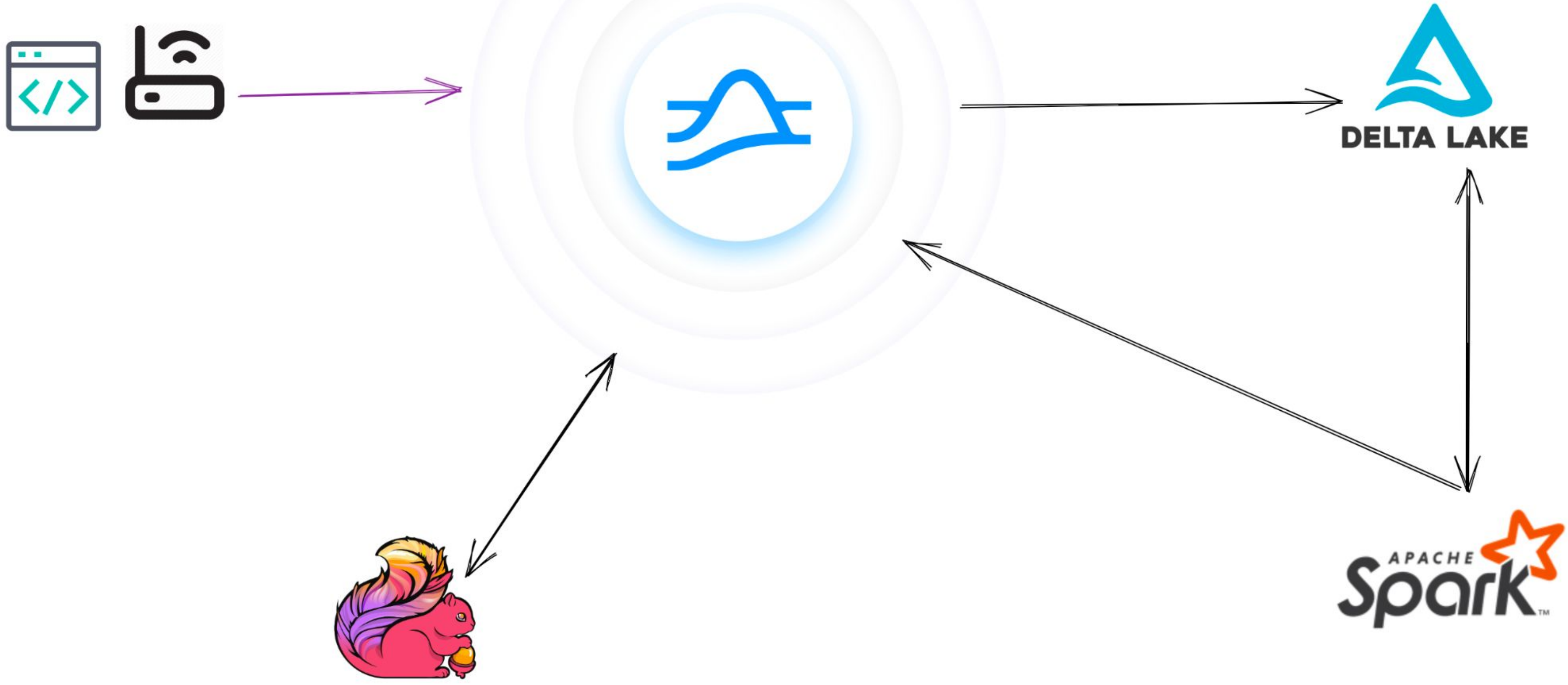
StreamNative

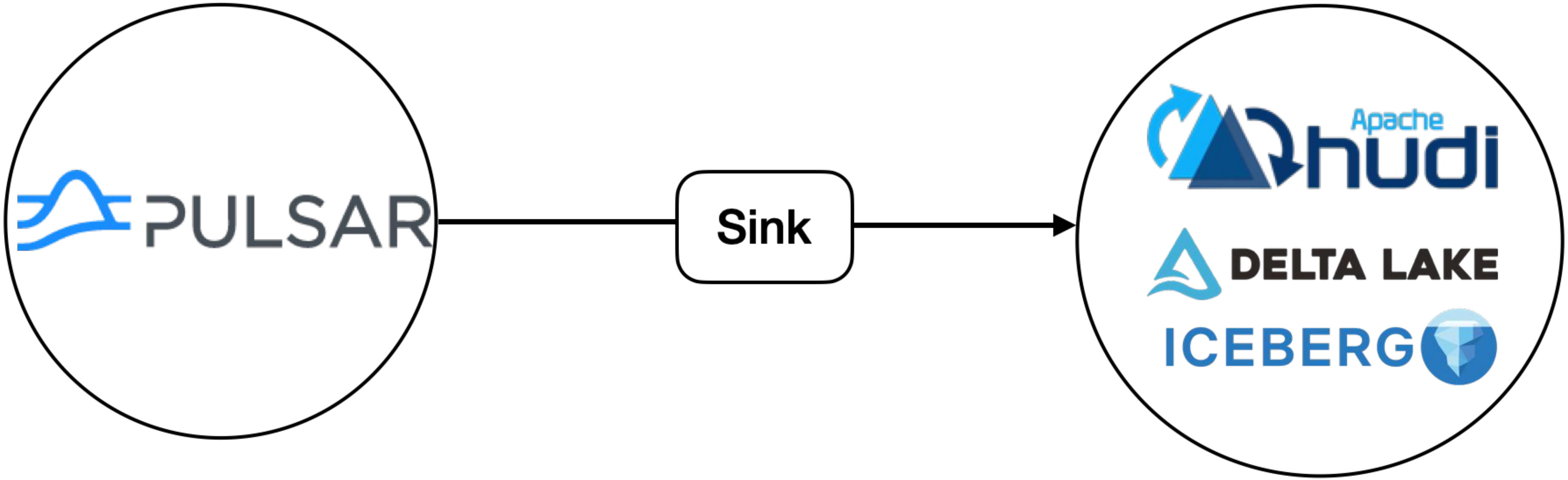
Pulsar Functions

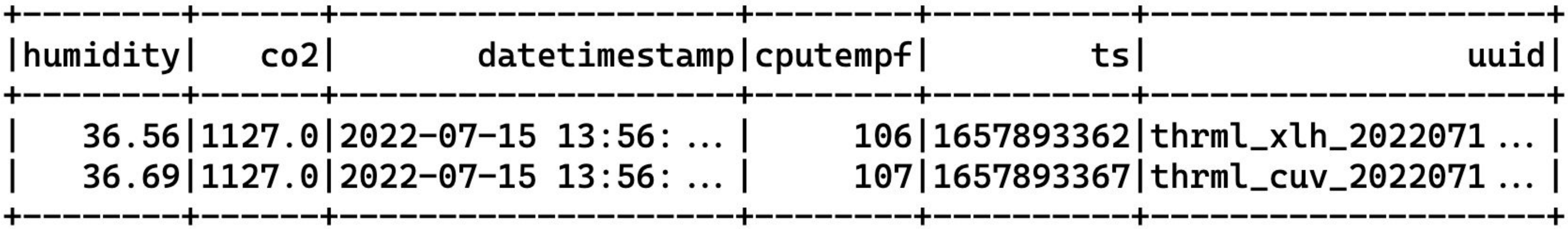
A serverless event streaming framework

- Lightweight computation similar to AWS Lambda.
- Specifically designed to use Apache Pulsar as a message bus.
- Function runtime can be located within Pulsar Broker.

"temperature": 28.238,
"humidity": 29.61,
"co2": 992.0

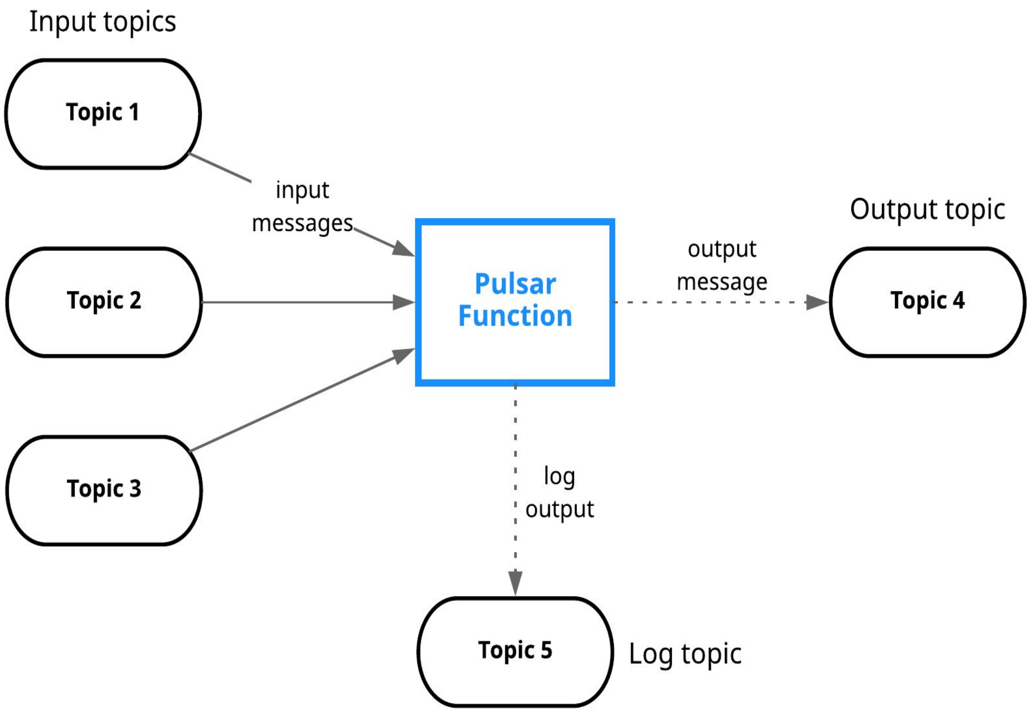






```
+---------+------+--------------------+--------+----------+--------------------+
|humidity|   co2|       datetimestamp|cputempf|        ts|                uuid|
+---------+------+--------------------+--------+----------+--------------------+
|    36.56|1127.0|2022-07-15 13:56:...|     106|1657893362|thrml_xlh_2022071...|
|    36.69|1127.0|2022-07-15 13:56:...|     107|1657893367|thrml_cuv_2022071...|
+---------+------+--------------------+--------+----------+--------------------+
```

Pulsar Functions



- Consume messages from one or more Pulsar topics.
- Apply user-supplied processing logic to each message.
- Publish the results of the computation to another topic.
- Support multiple programming languages (Java, Python, Go)
- Can leverage 3rd-party libraries to support the ***execution of ML models on the edge***.

Run a Local Standalone Bare Metal

```
wget
https://archive.apache.org/dist/pulsar/pulsar-2.10.1/apache-pulsar-2.10.1-bin.tar.gz

tar xvfz apache-pulsar-2.10.1-bin.tar.gz

cd apache-pulsar-2.10.1

bin/pulsar standalone

(For Pulsar SQL Support)

bin/pulsar sql-worker start
```

https://pulsar.apache.org/docs/en/standalone/

<or> Run in Docker

```
docker run -it \
  -p 6650:6650 \
  -p 8080:8080 \
  --mount source=pulsardata,target=/pulsar/data \
  --mount source=pulsarconf,target=/pulsar/conf \
  apachepulsar/pulsar:2.10.1 \
  bin/pulsar standalone
```

https://pulsar.apache.org/docs/en/standalone-docker/

Building Tenant, Namespace, Topics

```
bin/pulsar-admin tenants create conf

bin/pulsar-admin namespaces create conf/apachecon

bin/pulsar-admin tenants list

bin/pulsar-admin namespaces list conf

bin/pulsar-admin topics create
persistent://conf/apachecon/first

bin/pulsar-admin topics list conf/apachecon
```

Install Python 3 Pulsar Client

```
pip3 install pulsar-client=='2.10.1[all]'
```

```
Includes AARCH64, ARM, M2, INTEL, ...
```

For Python on Pulsar on Pi https://github.com/tspannhw/PulsarOnRaspberryPi

https://pypi.org/project/pulsar-client/2.10.0/#files

https://pulsar.apache.org/docs/en/client-libraries-python/

Building a Python 3 Producer

```
import pulsar

client = pulsar.Client('pulsar://localhost:6650')
producer
client.create_producer('persistent://conf/apachecon/first')
producer.send(('Simple Text Message').encode('utf-8'))
client.close()
```

Building a Python 3 Cloud Producer Oath

```
python3 prod.py -su pulsar+ssl://name1.name2.snio.cloud:6651 -t
persistent://public/default/pyth --auth-params
'{"issuer_url":"https://auth.streamnative.cloud", "private_key":"my.json",
"audience":"urn:sn:pulsar:name:myclustr"}'

from pulsar import Client, AuthenticationOauth2
parse = argparse.ArgumentParser(prog=prod.py')
parse.add_argument('-su', '--service-url', dest='service_url', type=str,
required=True)
args = parse.parse_args()
client = pulsar.Client(args.service_url,
          authentication=AuthenticationOauth2(args.auth_params))
```

https://github.com/streamnative/examples/blob/master/cloud/python/OAuth2Producer.py

StreamNative

https://github.com/tspannhw/FLiP-Pi-BreakoutGarden

Example Avro Schema Usage

```
import pulsar
from pulsar.schema import *
from pulsar.schema import AvroSchema
class thermal(Record):
    uuid = String()
client = pulsar.Client('pulsar://pulsar1:6650')
thermalschema = AvroSchema(thermal)
producer =
client.create_producer(topic='persistent://public/default/pi-thermal-avro',
         schema=thermalschema,properties={"producer-name": "thrm" })
thermalRec = thermal()
thermalRec.uuid = "unique-name"
producer.send(thermalRec,partition_key=uniqueid)
```

https://github.com/tspannhw/FLiP-Pi-Thermal

Example Json Schema Usage

```
import pulsar
from pulsar.schema import *
from pulsar.schema import JsonSchema
class weather(Record):
    uuid = String()
client = pulsar.Client('pulsar://pulsar1:6650')
wsc = JsonSchema(thermal)
producer =
client.create_producer(topic='persistent://public/default/wthr,schema=wsc,pro
perties={"producer-name": "wthr" })
weatherRec = weather()
weatherRec.uuid = "unique-name"
producer.send(weatherRec,partition_key=uniqueid)
```

https://github.com/tspannhw/FLiP-PulsarDevPython101

https://github.com/tspannhw/FLiP-Pi-Weather

Building a Python3 Consumer

```
import pulsar
client = pulsar.Client('pulsar://localhost:6650')
consumer =
client.subscribe('persistent://public/default/apachecon',subscription_name
='mine')

while True:
    msg = consumer.receive()
    print("Received message: '%s'" % msg.data())
    consumer.acknowledge(msg)
client.close()
```

MQTT from Python

```
pip3 install paho-mqtt

import paho.mqtt.client as mqtt
client = mqtt.Client("rpi4iot")
row = { }
row['gasKO'] = str(readings)
json_string = json.dumps(row)
json_string = json_string.strip()
client.connect("pulsar-server.com", 1883, 180)
client.publish("persistent://public/default/mqtt-2",
payload=json_string,qos=0,retain=True)
```

https://www.slideshare.net/bunkertor/data-minutes-2-apache-pulsar-with-mqtt-for-edge-computing-lightning-2022

Web Sockets from Python

```
pip3 install websocket-client

import websocket, base64, json
topic = 'ws://server:8080/ws/v2/producer/persistent/public/default/topic1'
ws = websocket.create_connection(topic)
message = "Hello Philly ETE Conference"
message_bytes = message.encode('ascii')
base64_bytes = base64.b64encode(message_bytes)
base64_message = base64_bytes.decode('ascii')
ws.send(json.dumps({'payload' : base64_message,'properties': {'device' :
'macbook'},'context' : 5}))
response =  json.loads(ws.recv())
```

https://github.com/tspannhw/FLiP-IoT/blob/main/wsreader.py
https://github.com/tspannhw/FLiP-IoT/blob/main/wspulsar.py
https://pulsar.apache.org/docs/en/client-libraries-websocket/

Kafka from Python

```
pip3 install kafka-python

from kafka import KafkaProducer
from kafka.errors import KafkaError

row = { }
row['gasKO'] = str(readings)
json_string = json.dumps(row)
json_string = json_string.strip()

producer = KafkaProducer(bootstrap_servers='pulsar1:9092',retries=3)
producer.send('topic-kafka-1', json.dumps(row).encode('utf-8'))
producer.flush()
```

https://docs.streamnative.io/platform/v1.0.0/concepts/kop-concepts

https://github.com/streamnative/kop

Deploy Python Functions

```
bin/pulsar-admin functions create --auto-ack true --py py/src/sentiment.py
--classname "sentiment.Chat" --inputs "persistent://public/default/chat"
--log-topic "persistent://public/default/logs" --name Chat --output
"persistent://public/default/chatresult"
```

https://github.com/tspannhw/pulsar-pychat-function

Pulsar IO Function in Python 3.9+

```
from pulsar import Function
import json


class Chat(Function):
    def __init__(self):
        pass

    def process(self, input, context):
        logger = context.get_logger()

        msg_id = context.get_message_id()

        fields = json.loads(input)
```

https://github.com/tspannhw/pulsar-pychat-function

Building a Golang Pulsar App

```
go get -u "github.com/apache/pulsar-client-go/pulsar"

import (
    "log"
    "time"
    "github.com/apache/pulsar-client-go/pulsar"
)
func main() {
    client, err := pulsar.NewClient(pulsar.ClientOptions{
        URL: "pulsar://localhost:6650",OperationTimeout: 30 * time.Second,
        ConnectionTimeout: 30 * time.Second,
    })
    if err != nil {
        log.Fatalf("Could not instantiate Pulsar client: %v", err)
    }
    defer client.Close()
}
```

http://pulsar.apache.org/docs/en/client-libraries-go/

Pulsar Producer

```
import java.util.UUID;
import java.net.URL;
import org.apache.pulsar.client.api.Producer;
import org.apache.pulsar.client.api.ProducerBuilder;
import org.apache.pulsar.client.api.PulsarClient;
import org.apache.pulsar.client.api.MessageId;
import org.apache.pulsar.client.impl.auth.oauth2.AuthenticationFactoryOAuth2;

PulsarClient client = PulsarClient.builder()
     .serviceUrl(serviceUrl)
     .authentication(
           AuthenticationFactoryOAuth2.clientCredentials(
             new URL(issuerUrl), new URL(credentialsUrl.), audience))
           .build();
```

Spring RabbitMQ/AMQP Producer

```
rabbitTemplate.convertAndSend(topicName,
                   DataUtility.serializeToJSON(observation));
```

Spring MQTT Producer

```
MqttMessage mqttMessage = new MqttMessage();
mqttMessage.setPayload(DataUtility.serialize(payload));
mqttMessage.setQos(1);
mqttMessage.setRetained(true);
mqttClient.publish(topicName, mqttMessage);
```

Spring Kafka Producer

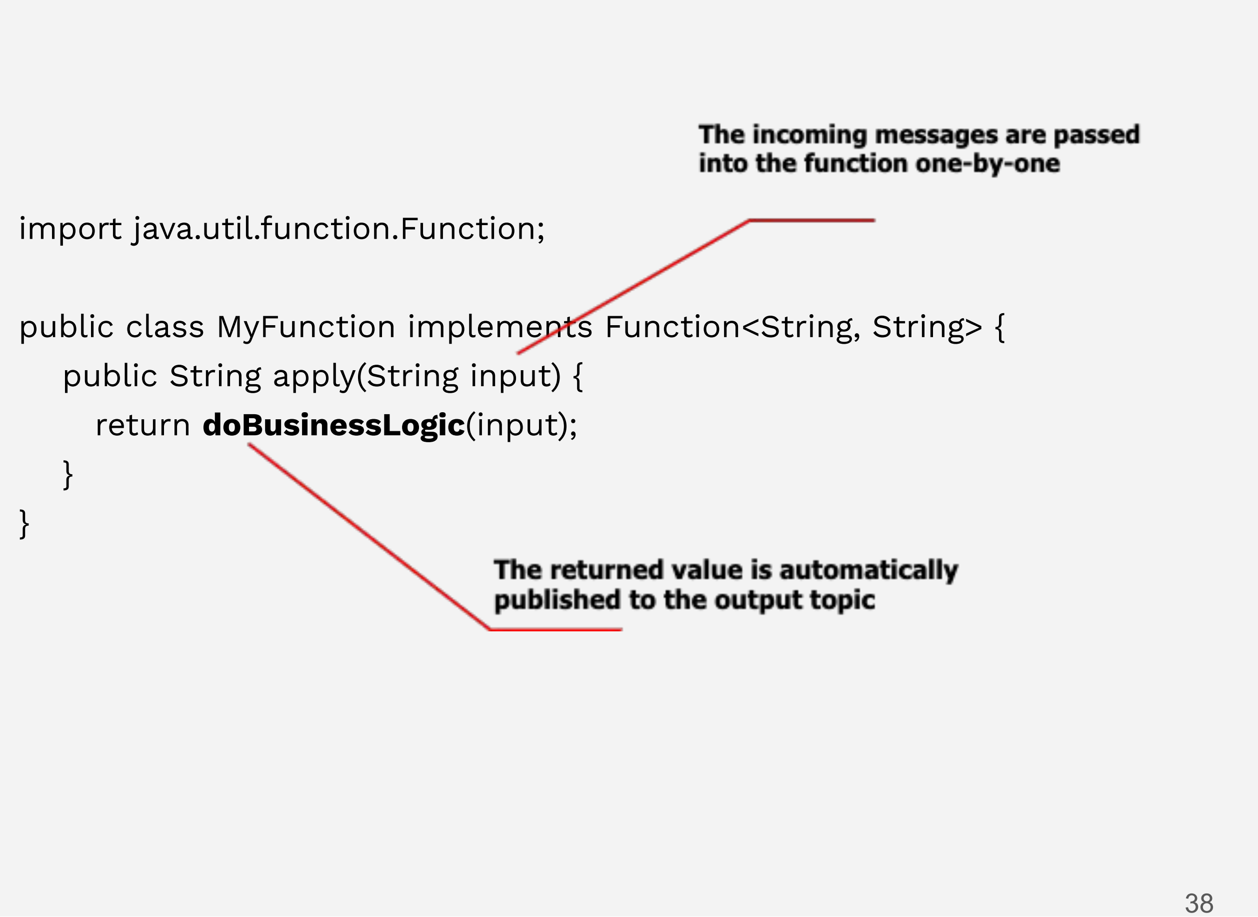
```
ProducerRecord<String, String> producerRecord = new
   ProducerRecord<>(topicName, uuidKey.toString(),
    DataUtility.serializeToJSON(message));
kafkaTemplate.send(producerRecord);
```

Pulsar Simple Producer

```
String pulsarKey = UUID.randomUUID().toString();
String OS = System.getProperty("os.name").toLowerCase();

ProducerBuilder<byte[]> producerBuilder = client.newProducer().topic(topic)
                .producerName("demo");
Producer<byte[]> producer = producerBuilder.create();

MessageId msgID = producer.newMessage().key(pulsarKey).value("msg".getBytes())
                   .property("device",OS).send();

producer.close();
client.close();
```

Pulsar Function Java

Your Code Here



```
import java.util.function.Function;

public class MyFunction implements Function<String, String> {
    public String apply(String input) {
       return doBusinessLogic(input);
    }
}
```

The incoming messages are passed into the function one-by-one

The returned value is automatically published to the output topic

Pulsar Function SDK

Your Code Here

```
import org.apache.pulsar.client.impl.schema.JSONSchema;
import org.apache.pulsar.functions.api.*;

public class AirQualityFunction implements Function<byte[], Void> {
  @Override
   public Void process(byte[] input, Context context) {
      context.getLogger().debug("File:” + new String(input));
      context.newOutputMessage(“topicname”,
                         JSONSchema.of(Observation.class))
                          .key(UUID.randomUUID().toString())
                          .property(“prop1”, “value1”)
                          .value(observation)
                          .send();
   }
}
```

Setting Subscription Type Java

```
Consumer<byte[]> consumer = pulsarClient.newConsumer()
              .topic(topic)
              .subscriptionName(“subscriptionName")

.subscriptionType(SubscriptionType.Shared)
              .subscribe();
```

Subscribing to a Topic and Setting Subscription Name Java

```
Consumer<byte[]> consumer = pulsarClient.newConsumer()
              .topic(topic)
              .subscriptionName(“subscriptionName")
              .subscribe();
```

Producing Object Events From Java

```
ProducerBuilder<Observation> producerBuilder =
pulsarClient.newProducer(JSONSchema.of(Observation.class))
             .topic(topicName)
             .producerName(producerName).sendTimeout(60,
                                         TimeUnit.SECONDS);
Producer<Observation> producer = producerBuilder.create();

msgID = producer.newMessage()
       .key(someUniqueKey)
       .value(observation)
       .send();
```

Monitoring and Metrics Check

```
curl http://pulsar1:8080/admin/v2/persistent/conf/europe/first/stats |
python3 -m json.tool
```

```
bin/pulsar-admin topics stats-internal persistent://conf/europe/first
```

```
curl http://pulsar1:8080/metrics/
```

```
bin/pulsar-admin topics stats-internal persistent://conf/europe/first
```

```
bin/pulsar-admin topics peek-messages --count 5 --subscription
ete-reader persistent://conf/europe/first
```

```
bin/pulsar-admin topics subscriptions persistent://conf/europe/first
```

Metrics: Broker

Broker metrics are exposed under "`/metrics`" at port **`8080`**.

You can change the port by updating `webServicePort` to a different port in the `broker.conf` configuration file.

All the metrics exposed by a broker are labeled with

`cluster=${pulsar_cluster}`.

The name of Pulsar cluster is the value of `${pulsar_cluster}`, configured in the `broker.conf` file.

For more information: https://pulsar.apache.org/docs/en/reference-metrics/#broker

Metrics: Broker

These metrics are available for brokers:

- Namespace metrics
  - Replication metrics
- Topic metrics
  - Replication metrics
- ManagedLedgerCache metrics
- ManagedLedger metrics
- LoadBalancing metrics
  - BundleUnloading metrics
  - BundleSplit metrics
- Subscription metrics
- Consumer metrics
- ManagedLedger bookie client metrics

Cleanup

```
bin/pulsar-admin topics delete persistent://conf/europe/first

bin/pulsar-admin namespaces delete conf/europe

bin/pulsar-admin tenants delete conf
```

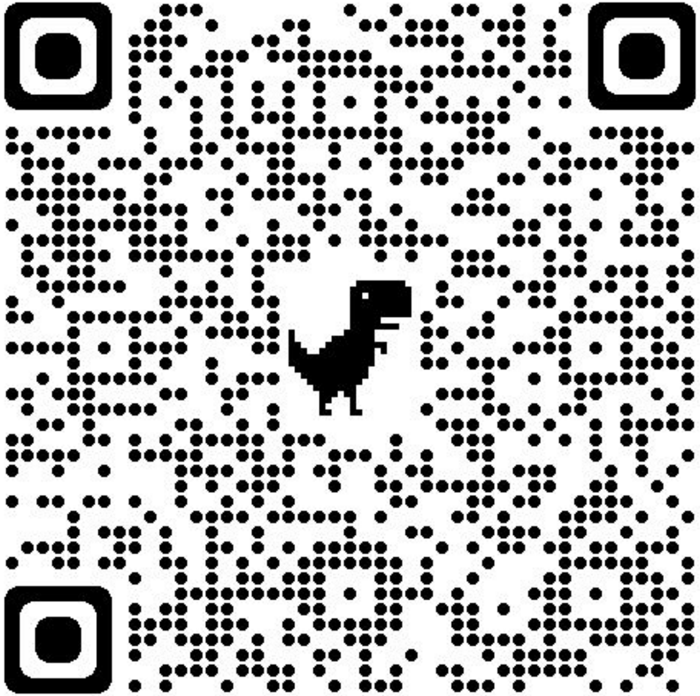
Java for Pulsar

- https://github.com/tspannhw/airquality
- https://github.com/tspannhw/FLiPN-AirQuality-REST
- https://github.com/tspannhw/pulsar-airquality-function
- https://github.com/tspannhw/FLiPN-DEVNEXUS-2022
- https://github.com/tspannhw/FLiP-Py-ADS-B
- https://github.com/tspannhw/pulsar-adsb-function
- https://github.com/tspannhw/airquality-amqp-consumer
- https://github.com/tspannhw/airquality-mqtt-consumer
- https://github.com/tspannhw/airquality-consumer
- https://github.com/tspannhw/airquality-kafka-consumer

Python For Pulsar on Pi

- https://github.com/tspannhw/FLiP-Pi-BreakoutGarden
- https://github.com/tspannhw/FLiP-Pi-Thermal
- https://github.com/tspannhw/FLiP-Pi-Weather
- https://github.com/tspannhw/FLiP-RP400
- https://github.com/tspannhw/FLiP-Py-Pi-GasThermal
- https://github.com/tspannhw/FLiP-PY-FakeDataPulsar
- https://github.com/tspannhw/FLiP-Py-Pi-EnviroPlus
- https://github.com/tspannhw/PythonPulsarExamples
- https://github.com/tspannhw/pulsar-pychat-function
- https://github.com/tspannhw/FLiP-PulsarDevPython101
- https://github.com/tspannhw/airquality

Get These Slides



Let's Keep in Touch!

Tim Spann
Developer Advocate

PaaSDev

https://www.linkedin.com/in/timothyspann

https://github.com/tspannhw