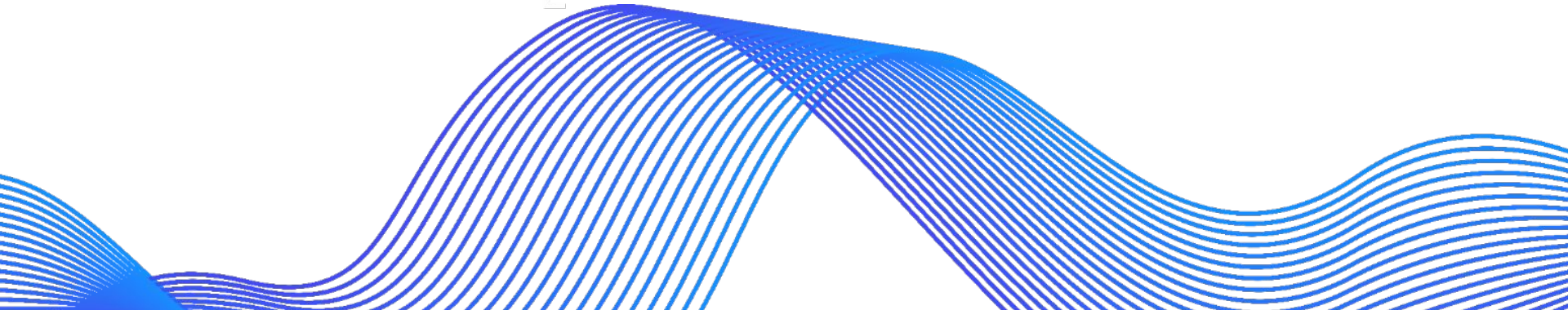
CODE ON THE BEACH

BROUGHT TO YOU BY Availity

StreamNative

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.

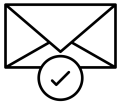
StreamNative

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

StreamNative

Why Apache Pulsar?

Unified
Messaging Platform



Guaranteed
Message Delivery

Resiliency

Infinite
Scalability

StreamNative

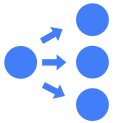
Pulsar Benefits

Building Microservices

Building Real Time Applications

Tiered storage

Asynchronous Communication



Highly Resilient

Key **Pulsar** Concepts: Architecture

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

PULSAR

Metadata & Service Discovery

Store Messages

MetaData Storage

etcd

- Stores metadata for both Pulsar and BookKeeper
- Service discovery

Apache BookKeeper™

Metadata & Service Discovery

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

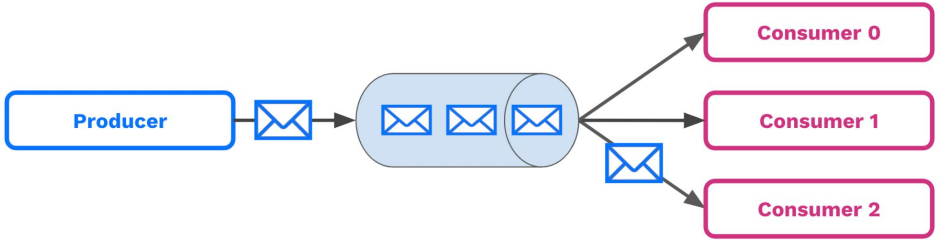
StreamNative

Messages - the basic unit of Pulsar

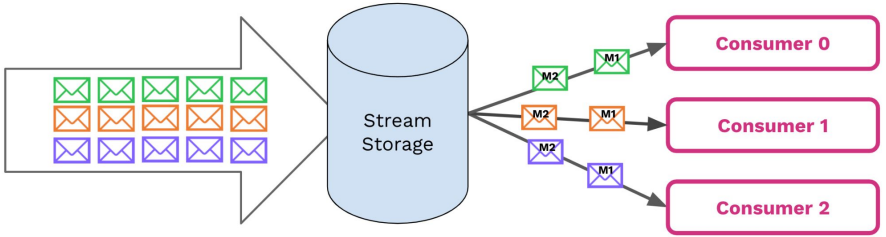
Component	Description
Value / data payload	The data carried by the message. All Pulsar messages contain raw bytes, although message data can also conform to data schemas.
Key	Messages are optionally tagged with keys, used in partitioning and also is useful for things like topic compaction.
Properties	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.

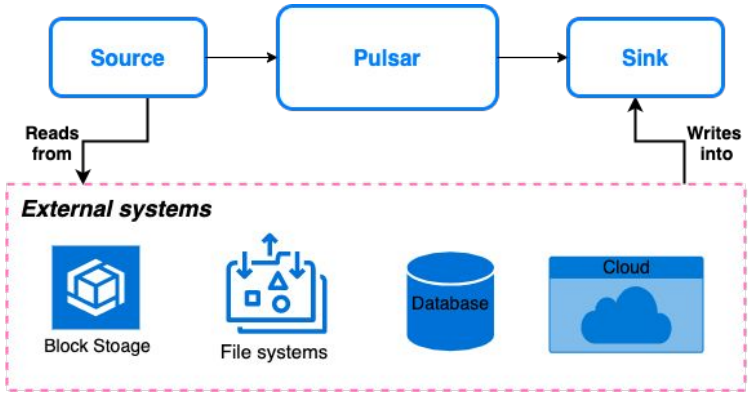


Streaming - Streaming works best in situations where the order of messages is important.



StreamNative

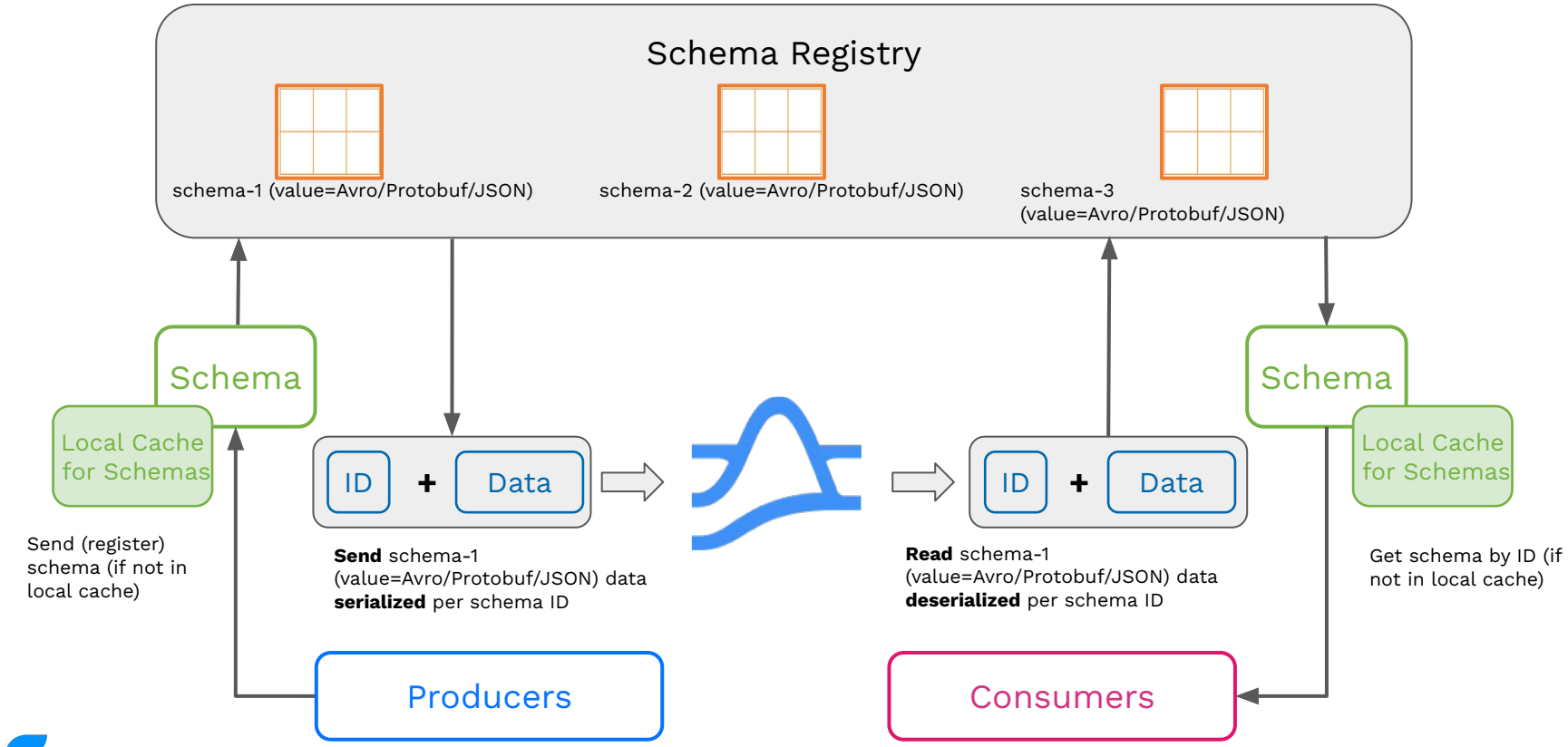
Connectivity



hub.streamnative.io

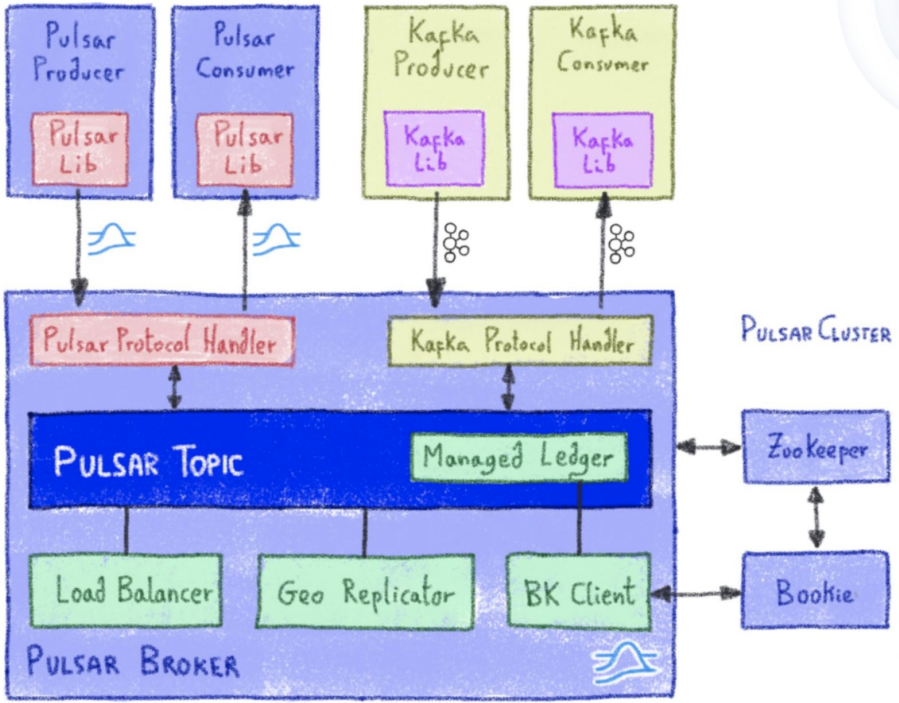
- **Functions** - Lightweight Stream Processing (Java, Python, Go)
- **Connectors** - Sources & Sinks (Cassandra, Kafka, …)
- **Protocol Handlers** - AoP (AMQP), KoP (Kafka), MoP (MQTT)
- **Processing Engines** - Flink, Spark, Presto/Trino via Pulsar SQL
- **Data Offloaders** - Tiered Storage - (S3)

Schema Registry



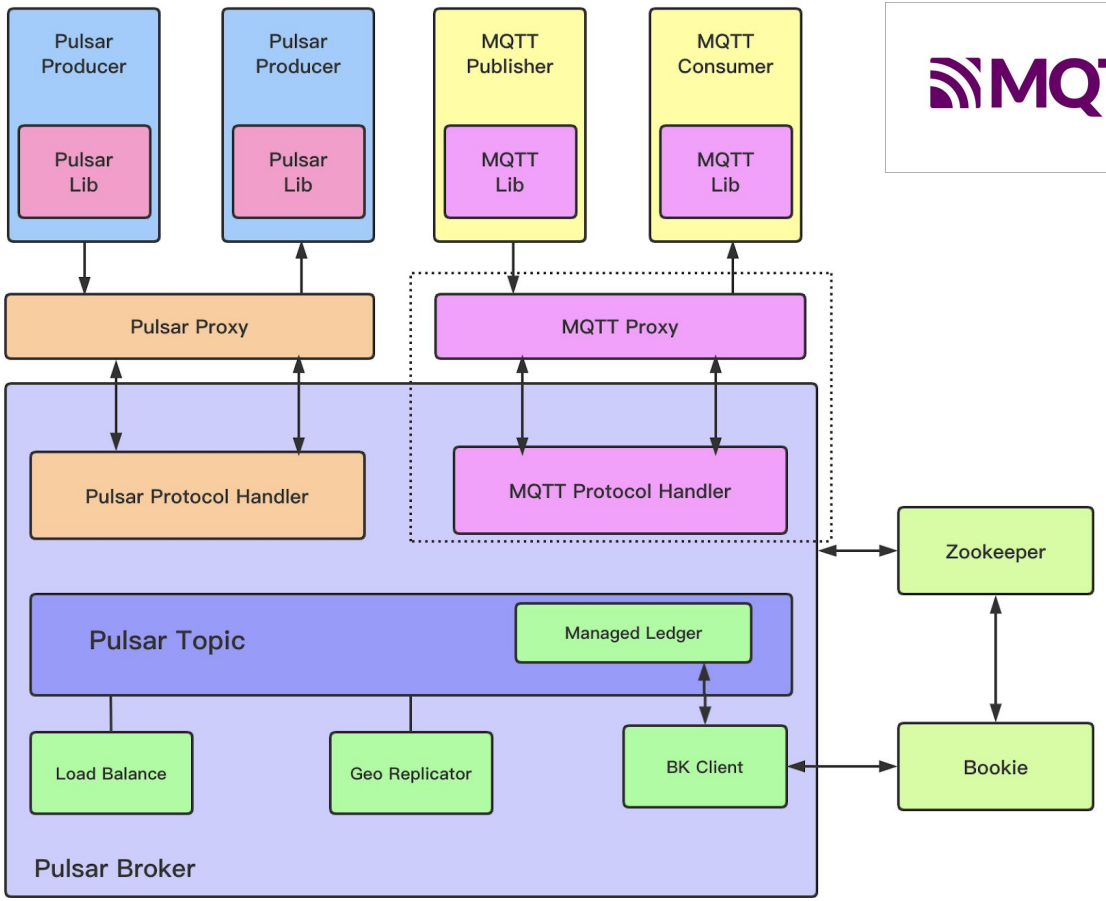
StreamNative

Kafka On Pulsar (KoP)



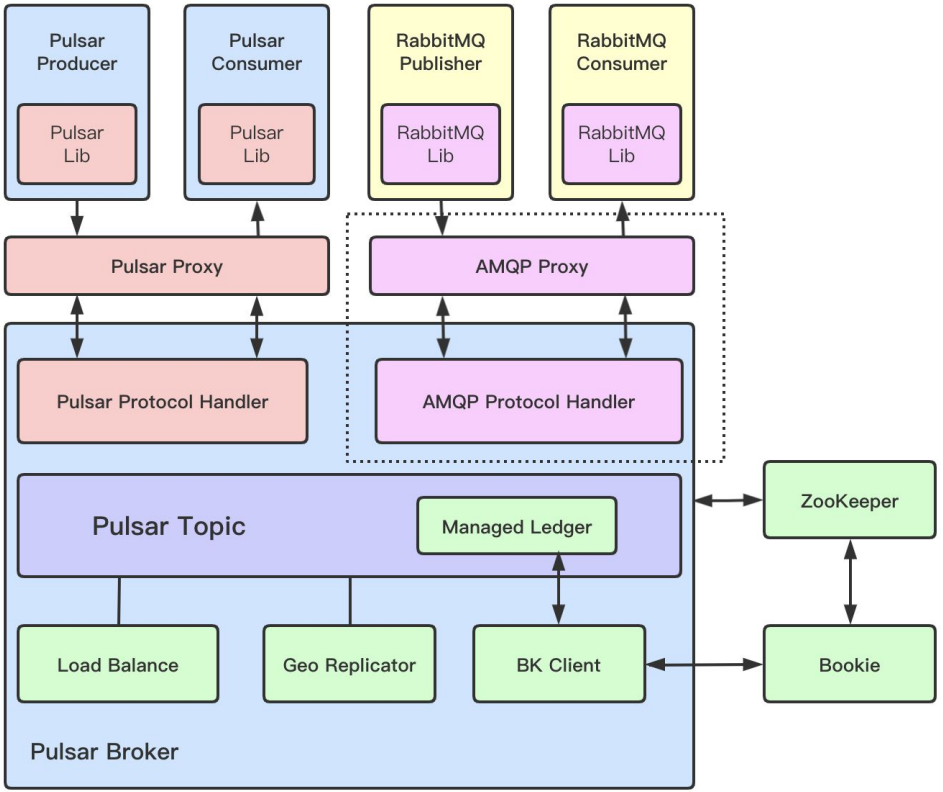
StreamNative

MQTT On Pulsar (MoP)



StreamNative

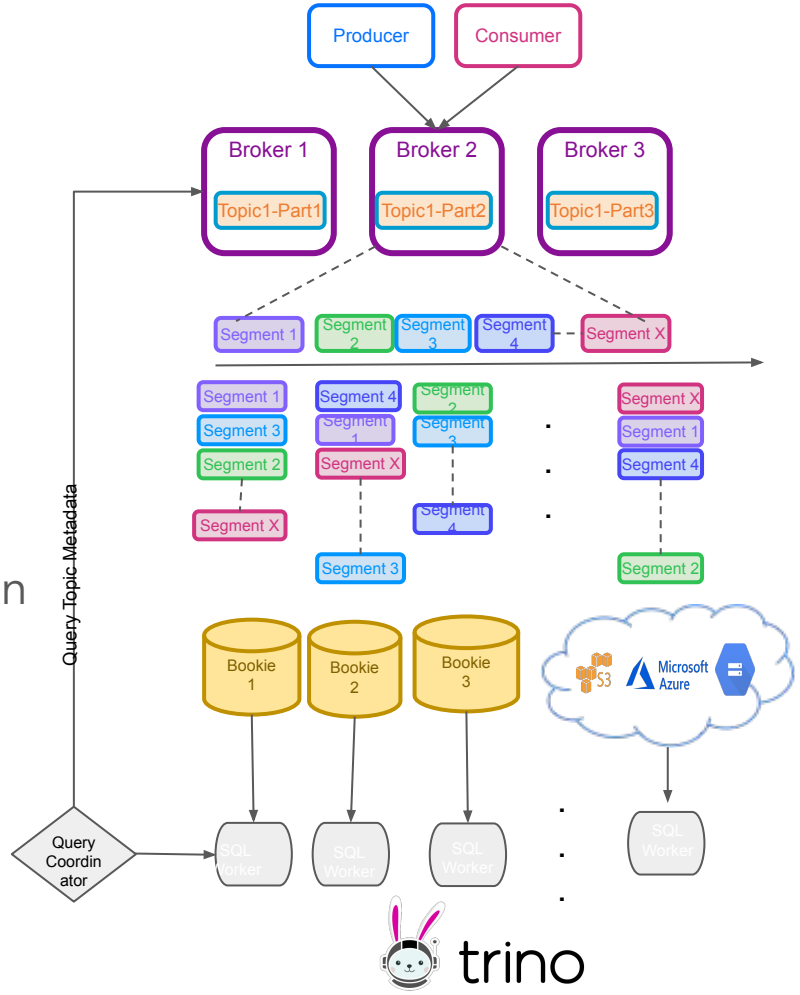
AMQP On Pulsar (AoP)



StreamNative

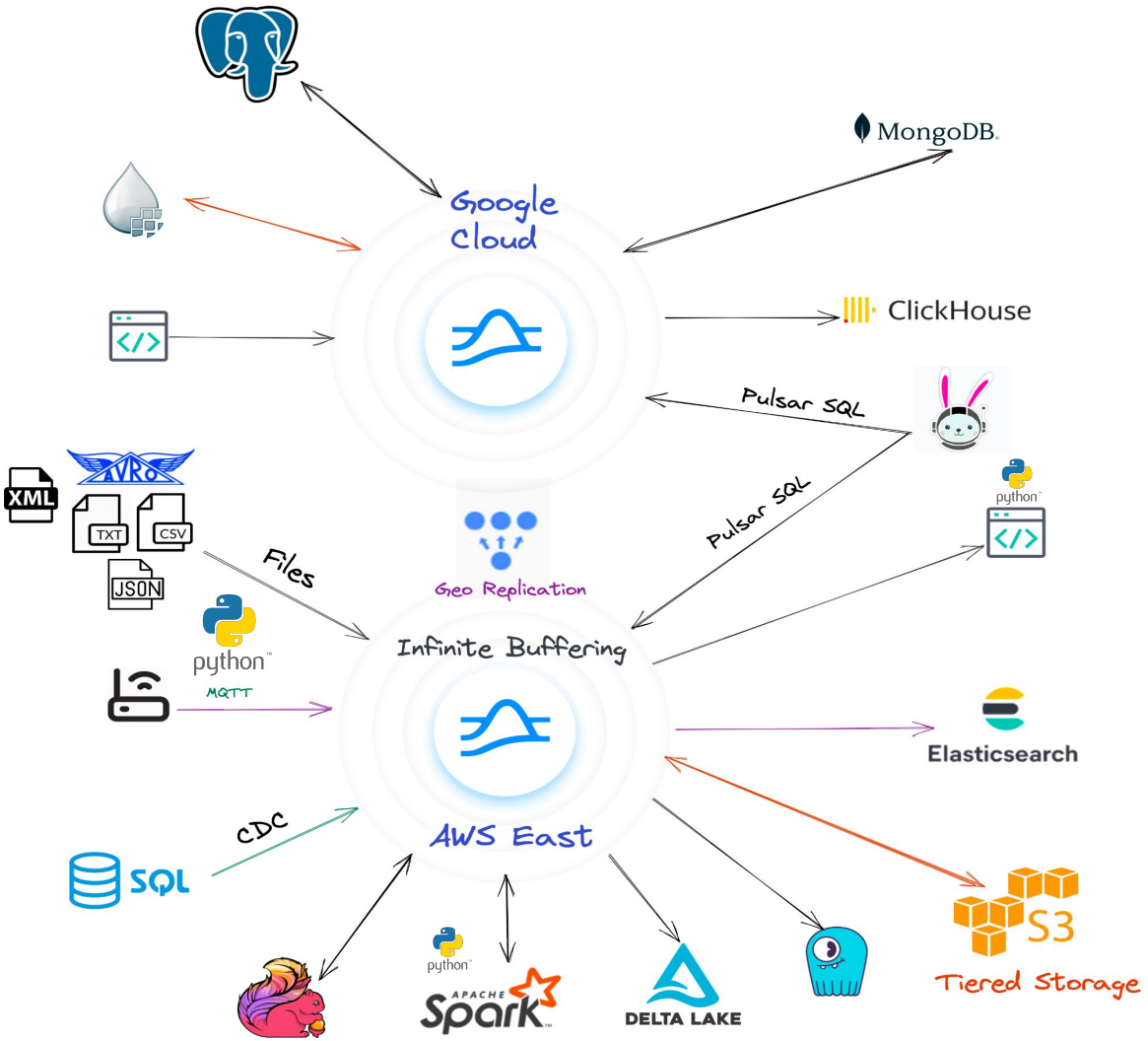
Pulsar SQL

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



StreamNative

- Buffer
- Batch
- Route
- Filter
- Aggregate
- Enrich
- Replicate
- Dedupe
- Decouple
- Distribute



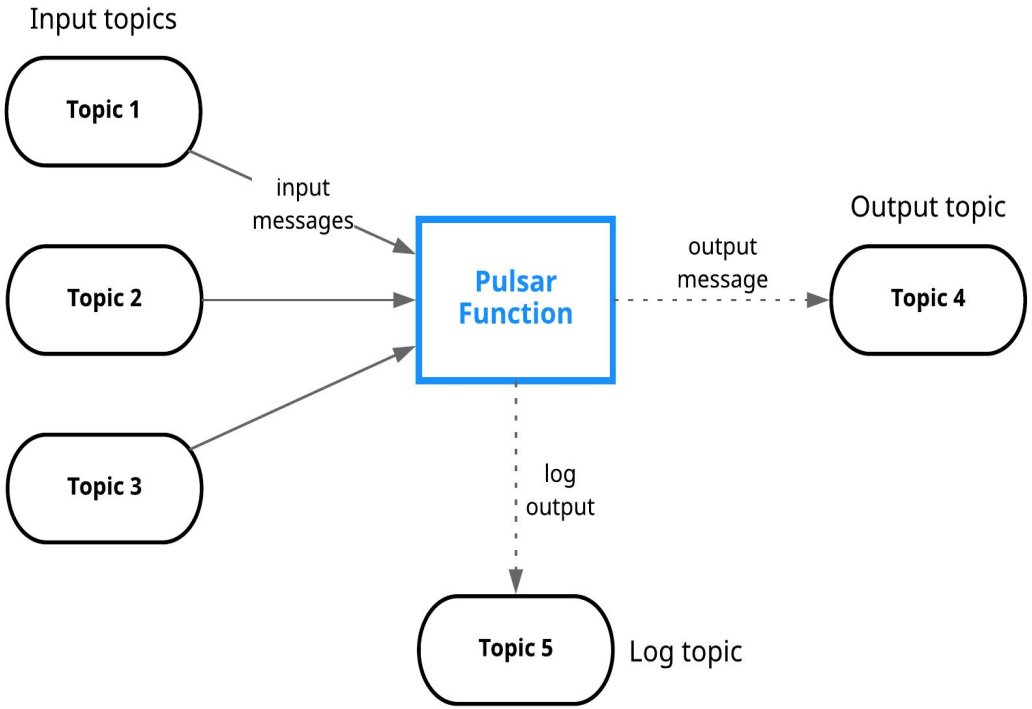
Pulsar Functions

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.

StreamNative

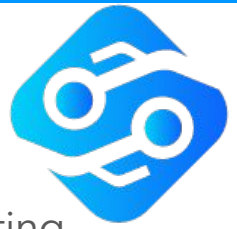
Pulsar Functions



- 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***.

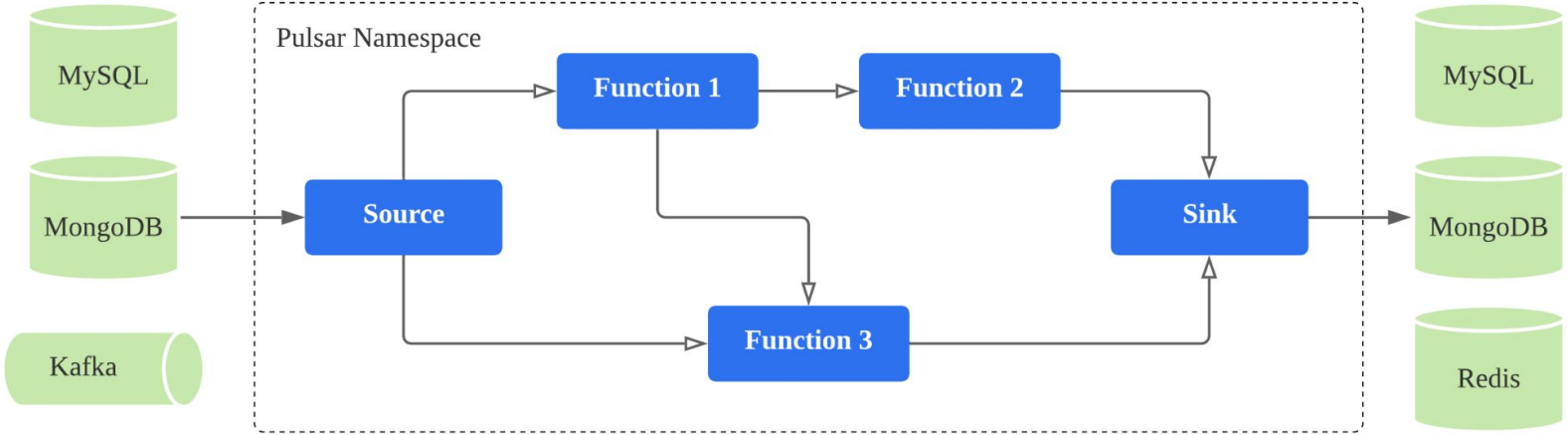
StreamNative

Function Mesh

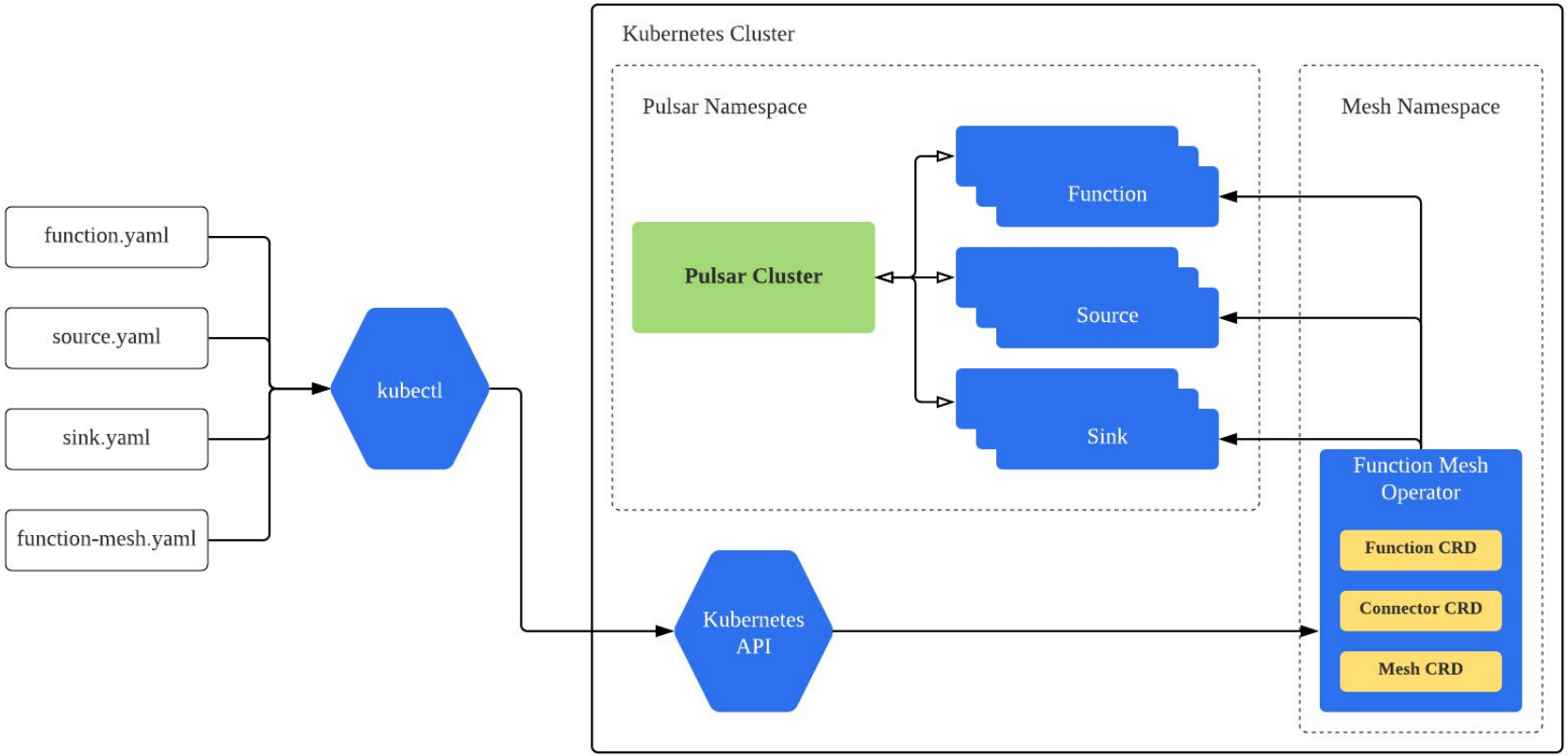


Pulsar Functions, along with Pulsar IO/Connectors, provide a powerful API for ingesting, transforming, and outputting data.

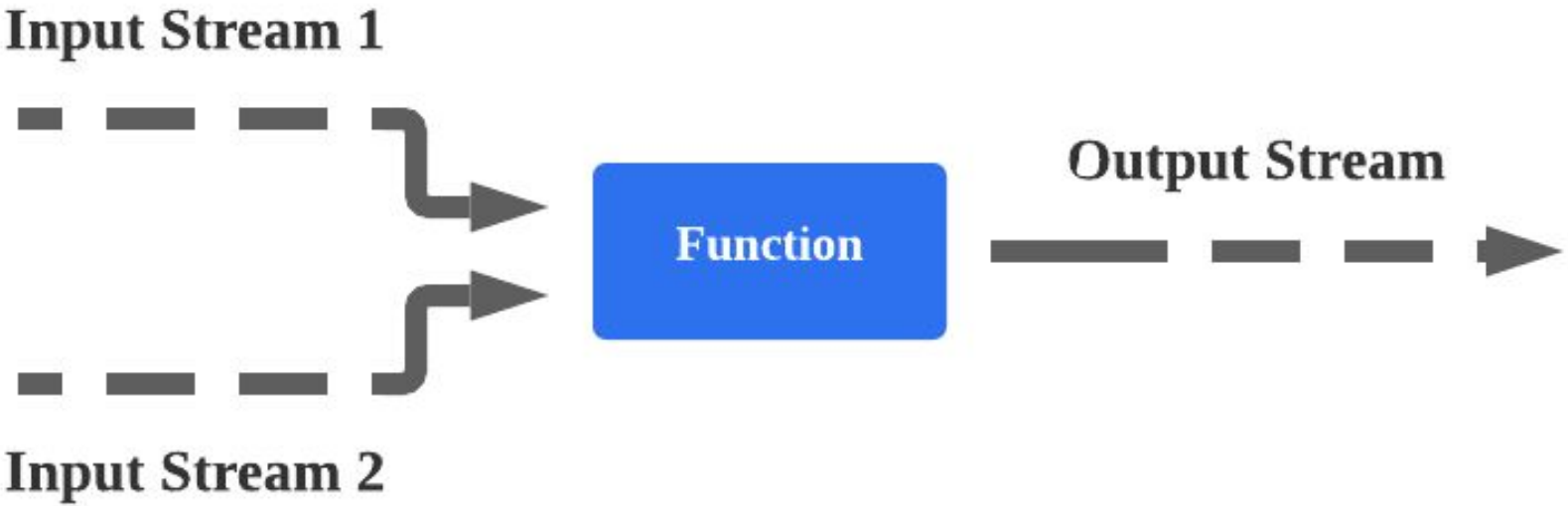
Function Mesh, another StreamNative project, makes it easier for developers to create entire applications built from sources, functions, and sinks all through a declarative API.



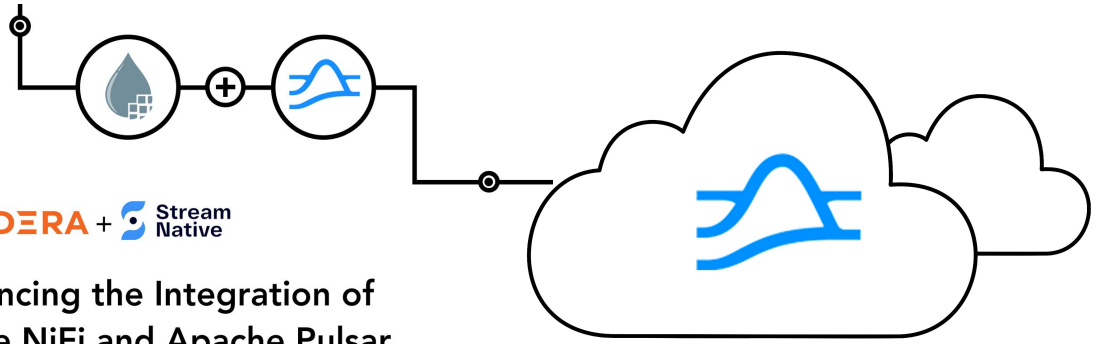
K8 Deploy

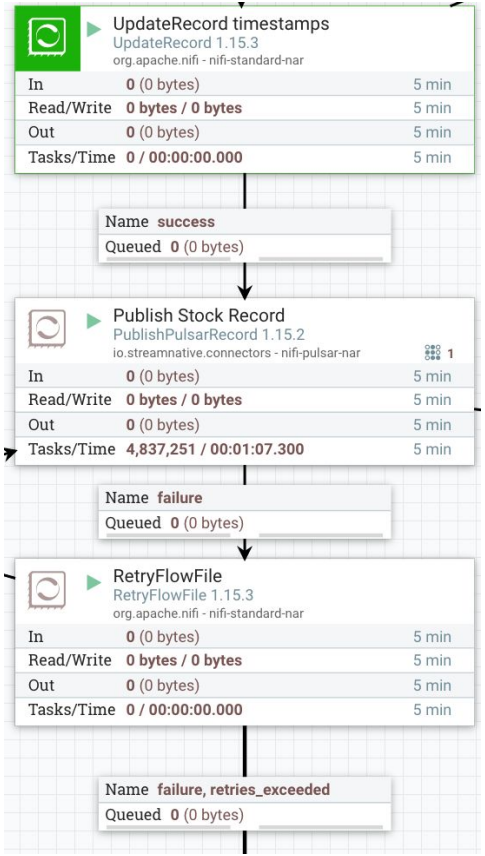

function.yaml
source.yaml
sink.yaml
function-mesh.yaml
kubectl
Kubernetes Cluster
Pulsar Namespace
Pulsar Cluster
Function
Source
Sink
Mesh Namespace
Function Mesh Operator
Function CRD
Connector CRD
Mesh CRD
Kubernetes API

Function Execution



Apache NiFi Pulsar Connector





https://streamnative.io/apache-nifi-connector/

StreamNative

Python 3 Coding

Code Along With Tim
<<DEMO>>

StreamNative

Building a Python3 Producer

```
import pulsar

client = pulsar.Client('pulsar://localhost:6650')
producer
client.create_producer('persistent://conf/ete/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

StreamNative

Building a Python3 Consumer

```
import pulsar
client = pulsar.Client('pulsar://localhost:6650')
consumer =
client.subscribe('persistent://conf/ete/first',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("rpi4-iot")
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/

StreamNative

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

StreamNative

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

```
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)
```

StreamNative

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()
}
```

StreamNative

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

Typed Java Client

```
Producer<User> producer = client.newProducer(Schema.AVRO(User.class)).create();
producer.newMessage()
        .value(User.builder()
                .userName("pulsar-user")
                .userId(1L)
                .build())
        .send();

Consumer<User> consumer =
client.newConsumer(Schema.AVRO(User.class)).create();
User user = consumer.receive();
```

Using Connection URLs

Java:

```
PulsarClient client = PulsarClient.builder()
    .serviceUrl("pulsar://broker1:6650")
    .build();
```

C#:

```
var client = new PulsarClientBuilder()
    .ServiceUrl("pulsar://broker1:6650")
    .Build();
```

Auth
C#

```
// Create the auth class
var tokenAuth = AuthenticationFactory.token("<jwttoken>");

// Pass the authentication implementation
var client = new PulsarClientBuilder()
     .ServiceUrl("pulsar://broker1:6650")
     .Authentication(tokenAuth)
     .Build();
```

Source Code

https://github.com/tspannhw/airquality

https://github.com/tspannhw/FLiPN-AirQuality-REST

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

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

https://github.com/tspannhw/FLiPN-DEVNEXUS-2022

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

StreamNative

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();
    }
}
```

StreamNative

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();
```

StreamNative

Building Pulsar SQL View

```
bin/pulsar sql


show catalogs;

show schemas in pulsar;

show tables in pulsar."conf/ete";

select * from pulsar."conf/ete"."first";

exit;
```

StreamNative

https://github.com/tspannhw/FLiP-Into-Trino

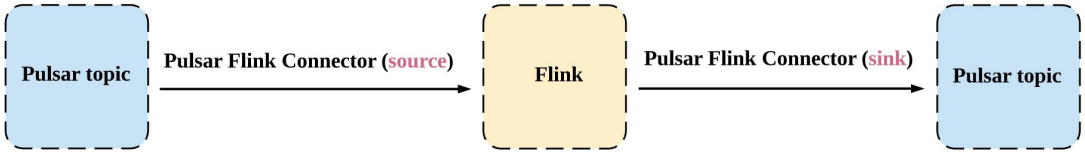
Spark + Pulsar

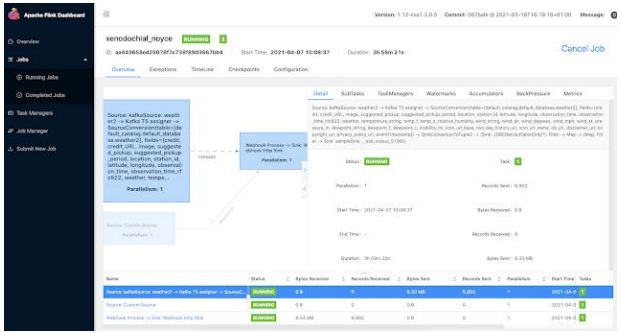

```
val dfPulsar = spark.readStream.format("pulsar")
               .option("service.url", "pulsar://pulsar1:6650")
               .option("admin.url", "http://pulsar1:8080")
               .option("topic", "persistent://public/default/airquality").load()


val pQuery = dfPulsar.selectExpr("*")
.writeStream.format("console")
.option("truncate", false).start()
```

https://pulsar.apache.org/docs/en/adaptors-spark/

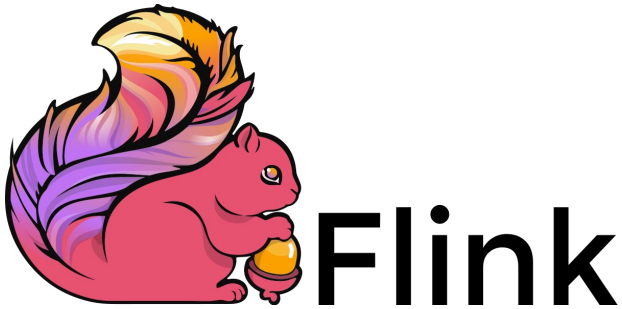

```
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 3.2.0
      /_/

Using Scala version 2.12.15
(OpenJDK 64-Bit Server VM, Java 11.0.11)
```

Apache Flink?





- Unified computing engine
- Batch processing is a special case of stream processing
- Stateful processing
- Massive Scalability
- Flink SQL for queries, inserts against Pulsar Topics
- Streaming Analytics
- Continuous SQL
- Continuous ETL
- Complex Event Processing
- Standard SQL Powered by Apache Calcite



StreamNative

Building Pulsar Flink SQL View

```
CREATE CATALOG pulsar WITH (
    'type' = 'pulsar',
    'service-url' = 'pulsar://pulsar1:6650',
    'admin-url' = 'http://pulsar1:8080',
    'format' = 'json'
);


select * from anytopicisatable;
```

StreamNative

https://github.com/tspannhw/FLiP-Into-Trino

Flink SQL


```
select aqi, parameterName, dateObserved, hourObserved, latitude,
longitude, localTimeZone, stateCode, reportingArea from
airquality;

select max(aqi) as MaxAQI, parameterName, reportingArea from
airquality group by parameterName, reportingArea;

select max(aqi) as MaxAQI, min(aqi) as MinAQI, avg(aqi) as
AvgAQI, count(aqi) as RowCount, parameterName, reportingArea
from airquality group by parameterName, reportingArea;
```

Flink SQL

```
                                SQL Query Result (Table)
Refresh: 1 s                       Page: Last of 1                    Updated: 15:52:48.189

                  uuid          ipaddress          cputempf          runtime
snr_20220323195238        192.168.1.229                99              453
snr_20220323195243        192.168.1.229               100              458

Q Quit                + Inc Refresh       G Goto Page       N Next Page       O Open Row
R Refresh             - Dec Refresh       L Last Page       P Prev Page
```

StreamNative

Building Spark SQL View

```
val dfPulsar = spark.readStream.format("pulsar")
      .option("service.url", "pulsar://pulsar1:6650")
      .option("admin.url", "http://pulsar1:8080")
      .option("topic", "persistent://public/default/pi-sensors")
      .load()

dfPulsar.printSchema()

val pQuery = dfPulsar.selectExpr("*")
             .writeStream.format("console")
             .option("truncate", false)
             .start()
```

StreamNative

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

Monitoring and Metrics Check

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

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

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

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

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

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

Cleanup

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

bin/pulsar-admin namespaces delete conf/ete

bin/pulsar-admin tenants delete conf
```

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.

These metrics are available for brokers:

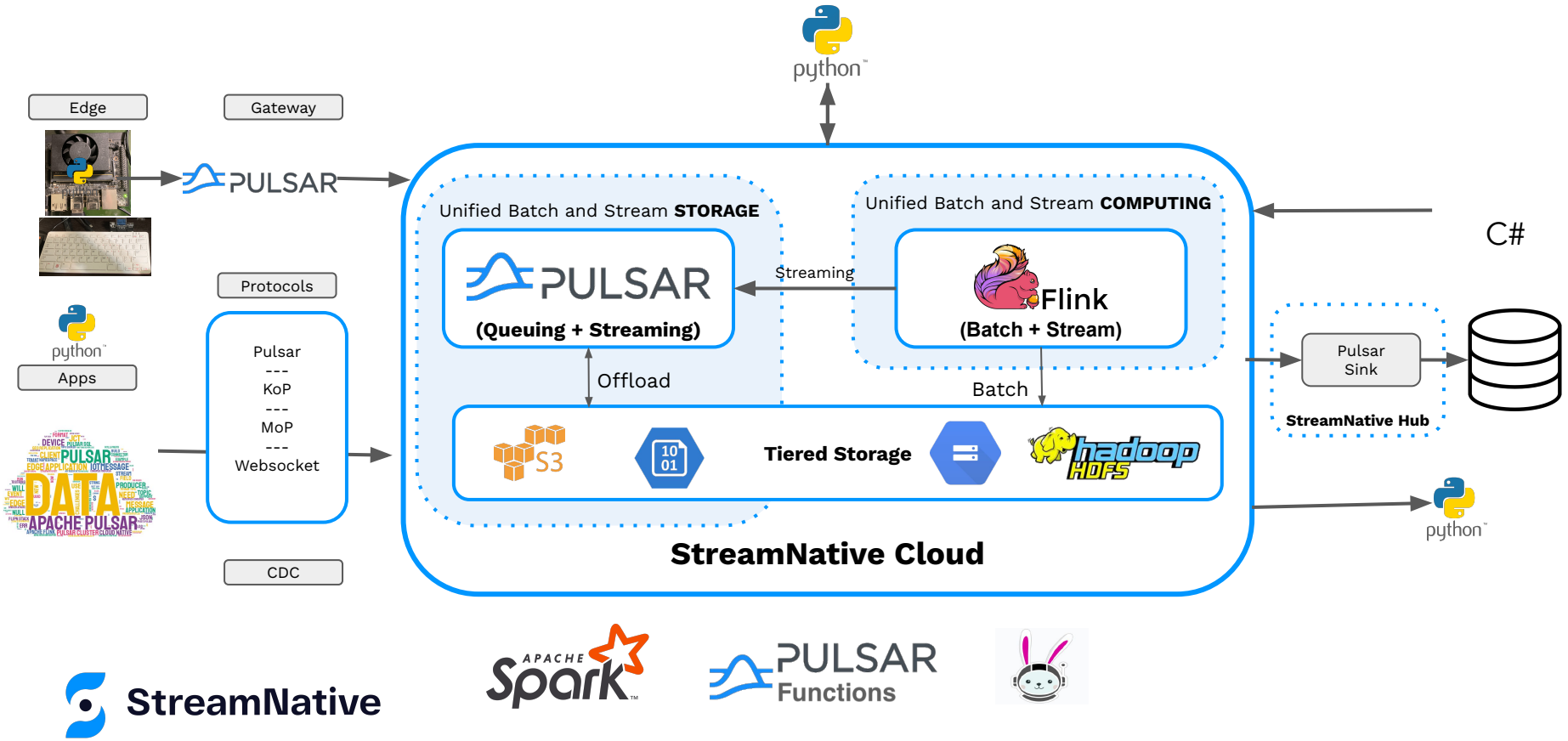
- Namespace metrics
  - Replication metrics
- Topic metrics
  - Replication metrics
- ManagedLedgerCache metrics
- ManagedLedger metrics
- LoadBalancing metrics
  - BundleUnloading metrics
  - BundleSplit metrics
- Subscription metrics
- Consumer metrics
- ManagedLedger bookie client metrics

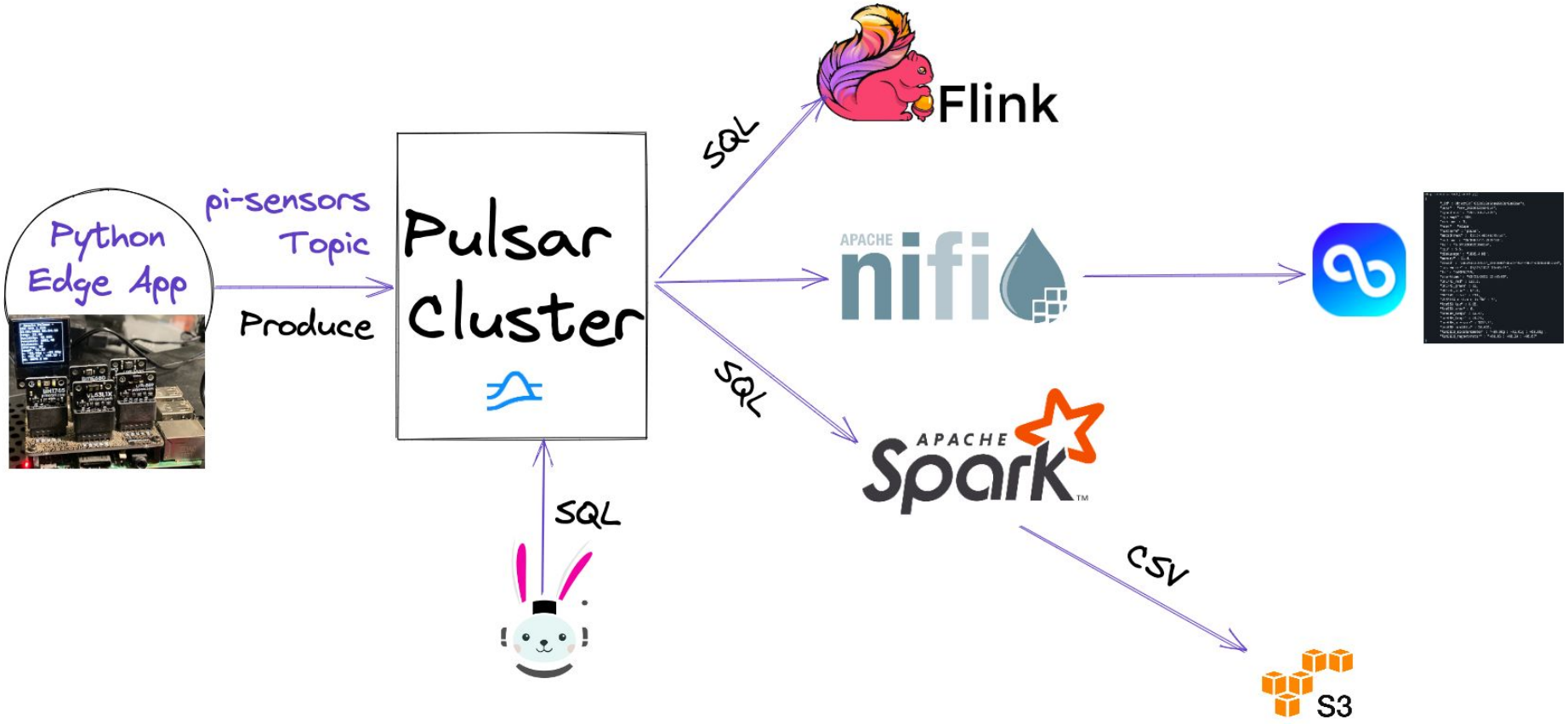
For more information: https://pulsar.apache.org/docs/en/reference-metrics/#broker

Use Cases

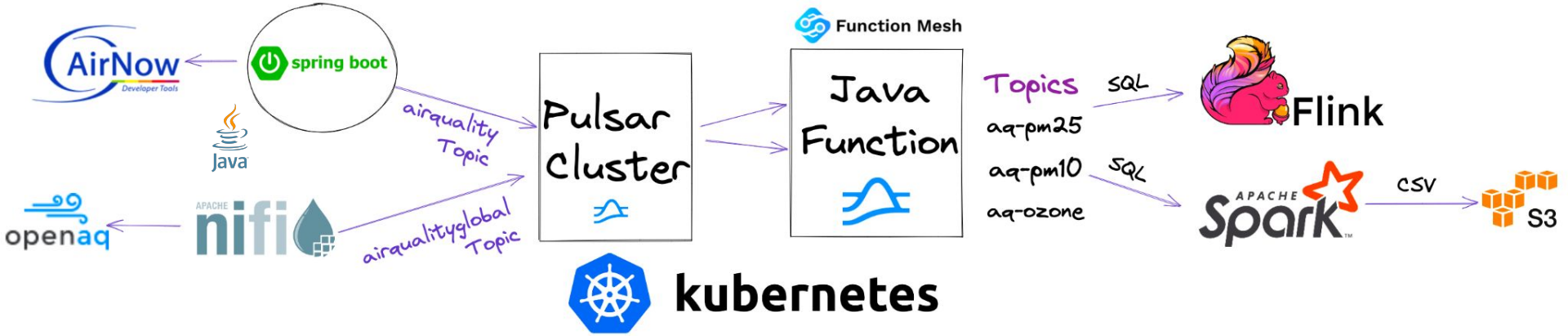
- Unified Messaging Platform
- AdTech
- Fraud Detection
- Connected Car
- IoT Analytics
- Microservices Development

Streaming FLiP-Py Apps





StreamNative



StreamNative

Pulsar Summit San Francisco

Hotel Nikko

August 18 2022

5 Keynotes

12 Breakout Sessions

1 Amazing Happy Hour

Save Your Spot Now

Use code CODEONTHEBEACH20 to get 20% off.

PULSAR SUMMIT

Hosted by Stream Native

Google Cloud

aws

ONEHOUSE

startree

intel

databricks

SCYLLA

yahoo!

beam

ververica

klaviyo

Flink

DELTA LAKE

Apache hudi

pinot

Pulsar Summit San Francisco Sponsorship Prospectus

Community Sponsorships Available

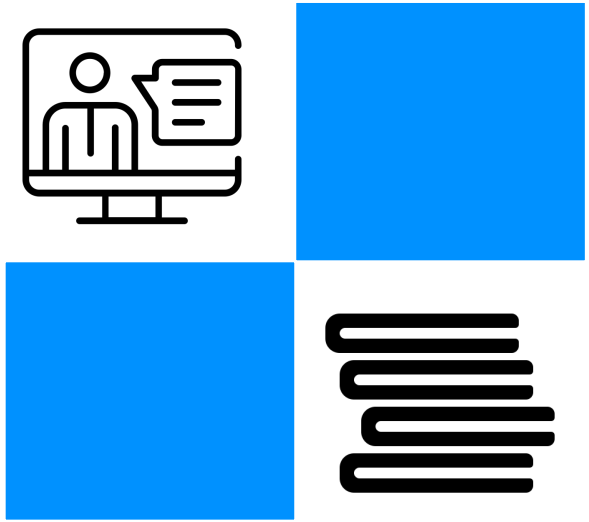
Help engage and connect the Apache Pulsar community by becoming an official sponsor for Pulsar Summit San Francisco 2022! Learn more about the requirements and benefits of becoming a community sponsor.

PULSAR SUMMIT

Hosted by StreamNative

RESOURCES

Here are resources to continue your journey with Apache Pulsar



StreamNative

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

Stream Native

Founded by the original developers of Apache Pulsar.

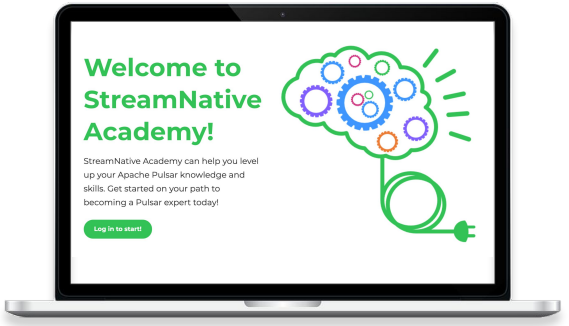
Passionate and dedicated team.

StreamNative helps teams to **capture**, **manage**, and **leverage data** using Pulsar's unified messaging and streaming platform.

streamnative.io

StreamNative Academy

Apache Pulsar Training



- Instructor-led courses
  - Pulsar Fundamentals
  - Pulsar Developers
  - Pulsar Operations
- On-demand learning with labs
- 300+ engineers, admins and architects trained!

Now Available

On-Demand Pulsar Training

Academy.StreamNative.io

StreamNative


PULSAR

Stream Native

Let's Keep in Touch!

Tim Spann

Developer Advocate

PaaSDev

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

https://github.com/tspannhw

StreamNative