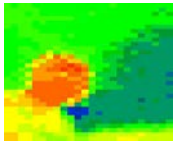
ML Enhanced Event Streaming Apps with Python Microservices

Tim Spann
Developer Advocate



Tim Spann

Principal Developer Advocate

FLiP(N) Stack = Flink, Pulsar and NiFi Stack

Streaming Systems & Data Architecture Expert

Experience:

- 15+ years of experience with streaming technologies including Pulsar, Flink, Spark, NiFi, Big Data, Cloud, MXNet, IoT, Python and more.
- Today, he helps to grow the Pulsar community sharing rich technical knowledge and experience at both global conferences and through individual conversations.

CLOUDERA

Pivotal

BARNES & NOBLE

Stream Native

EY

HORTONWORKS

Hewlett Packard Enterprise

airis.DATA

Description

In this talk, we will walk through how to build event streaming applications as functions running in with cloud-native messaging via Apache Pulsar that run near infinite scale in any cloud, docker, or K8. We will show you have to deploy ML functions to transform real-time data for IoT, Streaming Analytics, and many other use cases. After this talk, you will be able to build Python microservices with ease and deploy them anywhere utilizing the open-source unified streaming and messaging platform Apache Pulsar. Finally, we will show you have to add dashboards with Web Sockets, no code data sinks, integrate with Apache NiFi data pipelines, SQL Reports with Apache Spark, and finally, continuous ETL with Apache Flink. I have built many of these applications for many organizations as part of the FLiPN Stack. Let's build next-generation applications today regardless if your data is REST APIs, Sensors, Logs, NoSQL Sources, Events, or Database tables.

https://github.com/tspannhw?tab=repositories&q=FLiP&type=source

FLiP Stack Weekly

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

https://bit.ly/32dAJft

Building Real-Time Requires a Team

David Kjerrumgaard

- Apache Pulsar Committer | Author of Pulsar In Action
- Former Principal Software Engineer on Splunk's messaging team that is responsible for Splunk's internal Pulsar-as-a-Service platform.
- Former Director of Solution Architecture at Streamlio.

Stream Native

HORTONWORKS FedEx LOCKHEED MARTIN splunk>

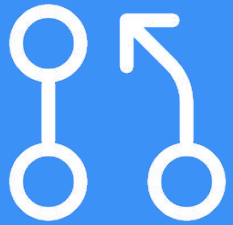
Apache Pulsar has a vibrant community

560+

Contributors

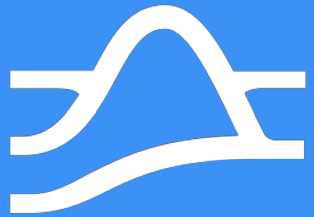
7,000+

Slack Members



10,000+

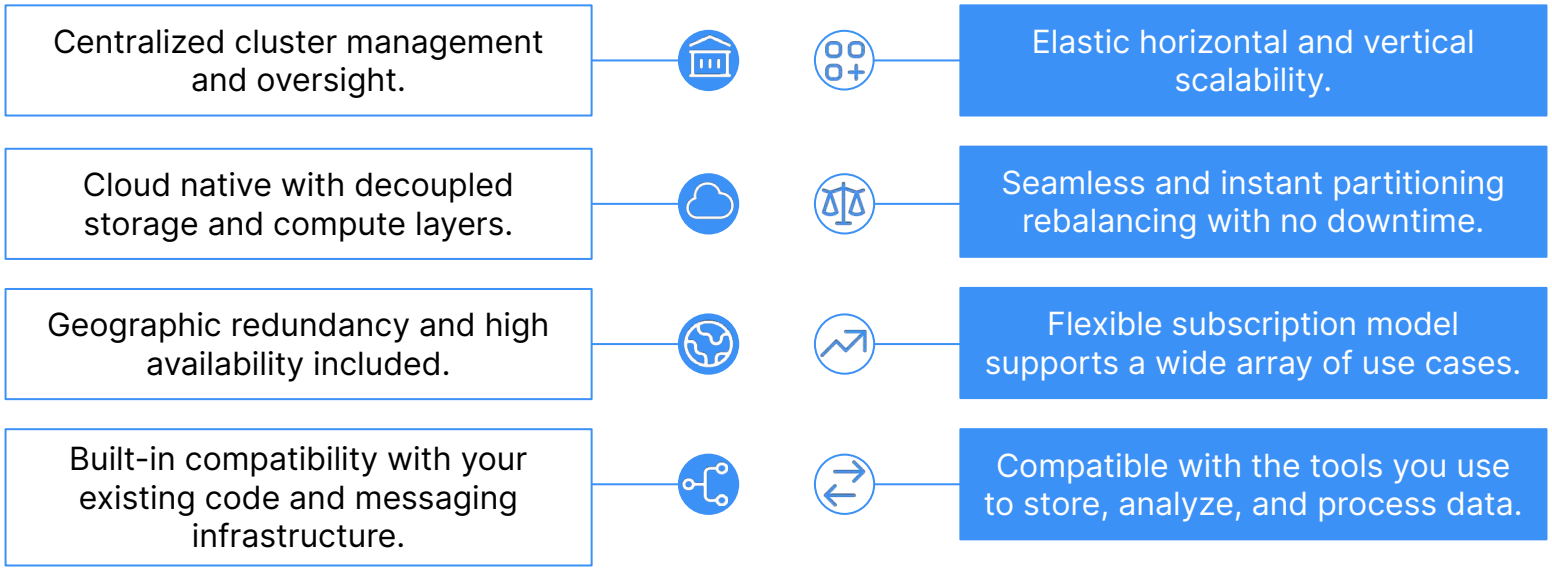
Commits



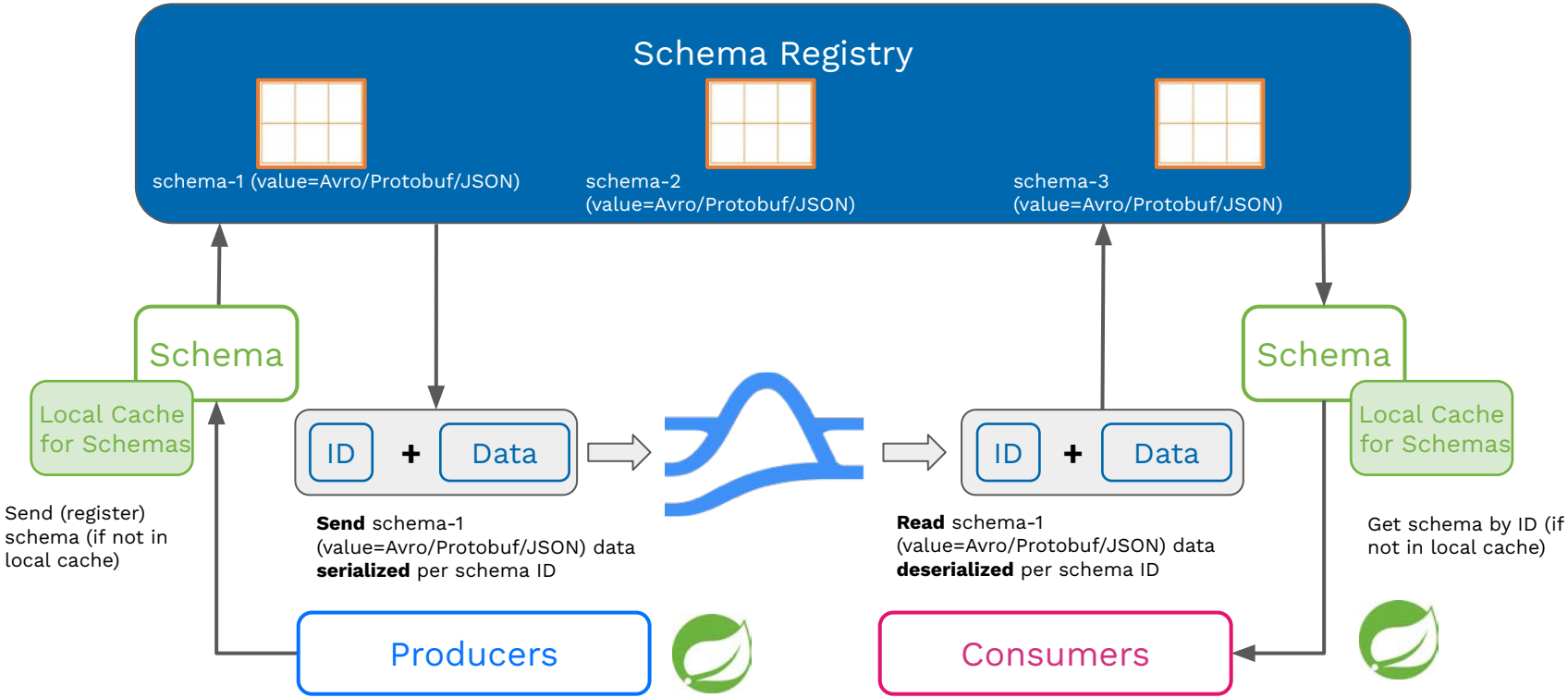
1,000+

Organizations Using Pulsar

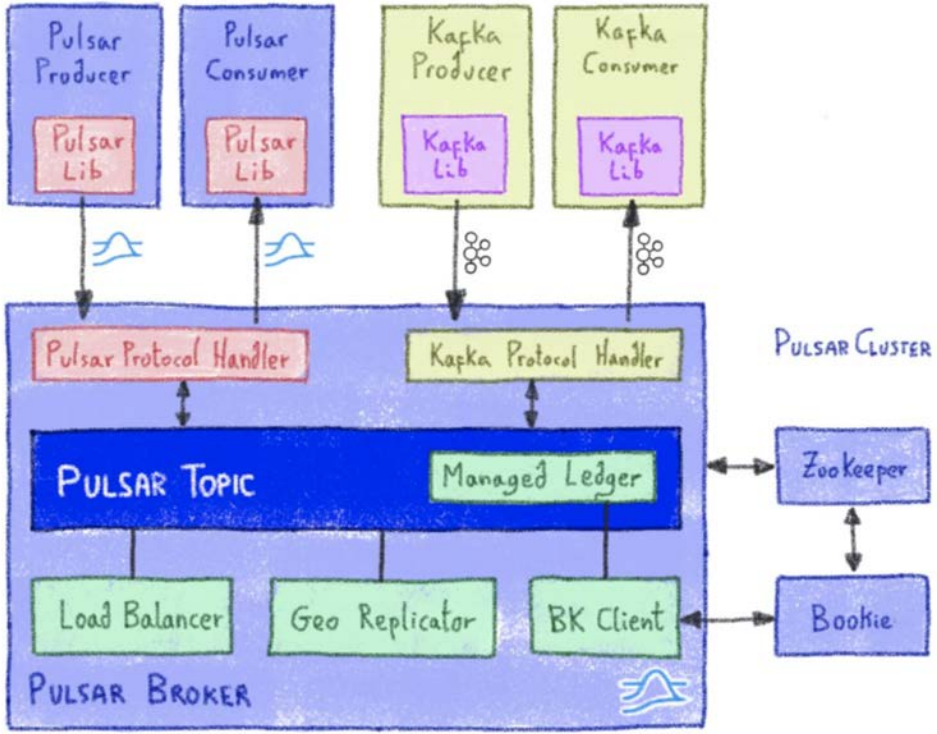
Pulsar Features



Integrated Schema Registry



Kafka on Pulsar (KoP)



Apache Pulsar Ecosystem

Protocol Handlers

kafka MQTT AMQP

Client Libraries

Scala WebSockets

Connectors (Sources & Sinks)

RabbitMQ kafka ACTIVEMQ amazon Kinesis debezium DELTA LAKE

Google Cloud Pub/Sub Google Big Query AWS Lambda ICEBERG hudi cassandra ... and more!

Pulsar Functions (Lightweight Stream Processing)

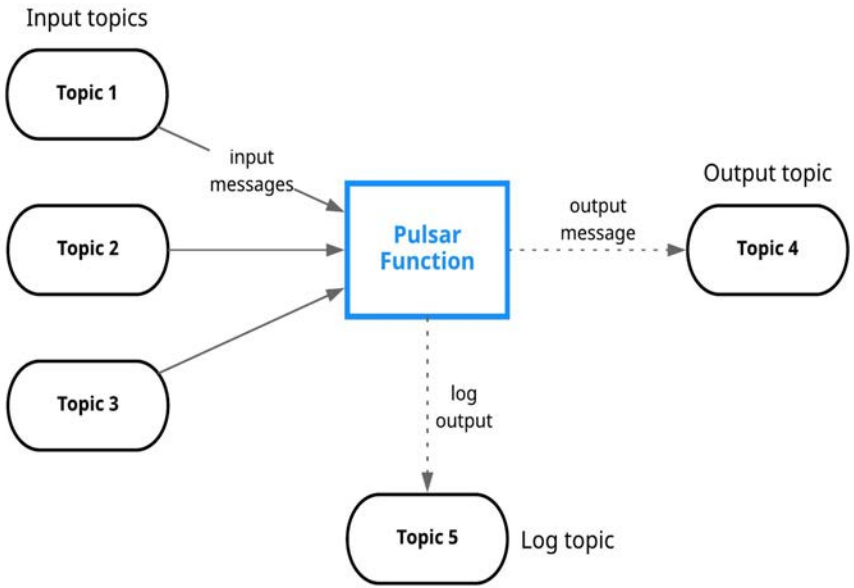
Processing Engines

Flink Spark presto trino AWS Lambda ... and more!

Data Offloaders (Tiered Storage)

Amazon S3 Google Cloud Storage hadoop

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.**

ML Function

Entire Function

```
from pulsar import Function
from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
import json
class Chat(Function):
    def __init__(self):
        pass
    def process(self, input, context):
        fields = json.loads(input)
        sid = SentimentIntensityAnalyzer()
        ss = sid.polarity_scores(fields["comment"])
        row = { }
        row['id'] = str(msg_id)
        if ss['compound'] < 0.00:
            row['sentiment'] = 'Negative'
        else:
            row['sentiment'] = 'Positive'
        row['comment'] = str(fields["comment"])
        json_string = json.dumps(row)
        return json_string
```

Install Python 3 Pulsar Client

```
pip3 install pulsar-client=='2.11.0[all]'

# Depending on Platform May Need C++ Client Built
```

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

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

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()
```

Producer with OAuth to Cloud

```
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

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 Python Producer

```
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()
```

Sending MQTT Messages

```
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

Sending Websocket Messages

```
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/

Sending Kafka Messages

```
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

DevOps: Deploying 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

X

Example Walk Through

```
import pulsar
from pulsar.schema import *

class Stock (Record):
    symbol = String()
    ts = Float()
    currentts = Float()
    volume = Float()
    price = Float()
    tradeconditions = String()
    uuid = String()
```

```
client = pulsar.Client('pulsar://localhost:6650')

producer = client.create_producer(topic='persistent://public/default/stocks'
,schema=JsonSchema(Stock),properties={"producer-name":
"py-stocks","producer-id": "pystocks1" })
```

```
uuid_key =
'{0}_{1}'.format(strftime("%Y%m%d%H%M%S",gmtime()),uuid.uuid4())
stockRecord = Stock()
stockRecord.symbol = stockitem['s']
stockRecord.ts = float(stockitem['t'])
stockRecord.currentts = float(strftime("%Y%m%d%H%M%S",gmtime()))
stockRecord.volume = float(stockitem['v'])
stockRecord.price = float(stockitem['p'])
stockRecord.tradeconditions = ','.join(stockitem['c'])
stockRecord.uuid = uuid_key


if ( stockitem['s'] != '' ):
    producer.send(stockRecord,partition_key=str(uuid_key))
```

StreamNative

Show 100 entries

Search:

Key	Publish Time	Service	Msg	Descr	Pub Date	System Time
95aa3284-26d6-44f2-8d37-be9460ebe6bc	2022-10-21T17:40:34.101-04:00	bus	BUS 1 - Feb 10, 2021 11:28:47 AM	Various Bus Routes: Long-Term Bus Stop Adjustments in Newark - Effective Immediately through February 2023	Feb 10, 2021 11:28:47 AM	1666388421788
37af063b-8665-488c-bee4-c82a51dba5e8	2022-10-21T17:40:34.081-04:00	bus	BUS 815 - May 07, 2021 11:14:34 AM	Bus Route Nos. 811, 814, 815 & 818: Temporary Bus Stop Changes in New Brunswick â€ Effective Immediately	May 07, 2021 11:14:34 AM	1666388421787
965c3b27-9904-420b-9752-99c38ca2e092	2022-10-21T17:40:34.061-04:00	bus	BUS 814 - May 07, 2021 11:14:34 AM	Bus Route Nos. 811, 814, 815 & 818: Temporary Bus Stop Changes in New Brunswick â€ Effective Immediately	May 07, 2021 11:14:34 AM	1666388421786
dc528367-7ddf-4059-8a2b-130f591660bb	2022-10-21T17:40:34.041-04:00	bus	BUS 199 - Jun 25, 2021 10:54:56 AM	Nos. 190 Series & 324 via PABT: Service Delays & Detours due to Major Passaic Co. Bridge Construction Project â€ Effective Immediately through Summer 2022	Jun 25, 2021 10:54:56 AM	1666388421786
ab20c0a9-6d08-4a70-b255-e84c0f3fd259	2022-10-21T17:40:34.021-04:00	bus	BUS 324 - Jun 25, 2021 10:54:56 AM	Nos. 190 Series & 324 via PABT: Service Delays & Detours due to Major Passaic Co. Bridge Construction Project â€ Effective Immediately through Summer 2022	Jun 25, 2021 10:54:56 AM	1666388421786
0f4d95c6-9e1d-48dc-84fe-bde22c9d04b8	2022-10-21T17:40:34.001-04:00	bus	BUS 197 - Jun 25, 2021 10:54:56 AM	Nos. 190 Series & 324 via PABT: Service Delays & Detours due to Major Passaic Co. Bridge Construction Project â€ Effective Immediately through Summer 2022	Jun 25, 2021 10:54:56 AM	1666388421785
c6a7f5ae-d524-49cf-babe-78be1bf07b94	2022-10-21T17:40:33.981-04:00	bus	BUS 195 - Jun 25, 2021 10:54:56 AM	Nos. 190 Series & 324 via PABT: Service Delays & Detours due to Major Passaic Co. Bridge Construction Project â€ Effective Immediately through Summer 2022	Jun 25, 2021 10:54:56 AM	1666388421784
df23b909-e329-4101-ae30-86566047fe74	2022-10-21T17:40:33.961-04:00	bus	BUS 194 - Jun 25, 2021 10:54:56 AM	Nos. 190 Series & 324 via PABT: Service Delays & Detours due to Major Passaic Co. Bridge Construction Project â€ Effective Immediately through Summer 2022	Jun 25, 2021 10:54:56 AM	1666388421784
39240318-13ae-4c65-a922-02ad57923b6c	2022-10-21T17:40:33.94-04:00	bus	BUS 190 - Jun 25, 2021 10:54:56 AM	Nos. 190 Series & 324 via PABT: Service Delays & Detours due to Major Passaic Co. Bridge Construction Project â€ Effective Immediately through Summer 2022	Jun 25, 2021 10:54:56 AM	1666388421783

Example Web Page

```
   <link rel="stylesheet" type="text/css"
href="https://cdn.datatables.net/1.11.3/css/jquery.dataTables.min.css">
    <script type="text/javascript" language="javascript"
src="https://code.jquery.com/jquery-3.5.1.js">
</script>
    <script type="text/javascript" language="javascript"
src="https://cdn.datatables.net/1.11.3/js/jquery.dataTables.min.js">
</script>
```

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

```
<table id="example" class="display" style="width:100%">
   <thead>
   <tr>
      <th><b>Key</b></th>
      <th><b>Publish Time</b></th>
      <th><b>Msg</b></th>
      <th><b>Latitude</b></th>
      <th><b>Longitude</b></th>
      <th><b>Pub Date</b></th>
      <th><b>System Time</b></th>  </tr>    </thead>
 <tfoot>
   <tr>
      <th><b>Key</b></th>
      <th><b>Publish Time</b></th>
      <th><b>Msg</b></th>
      <th><b>Latitude</b></th>
      <th><b>Longitude</b></th>
      <th><b>Pub Date</b></th>
      <th><b>System Time</b></th>
   </tr>
   </tfoot>
</table>
```

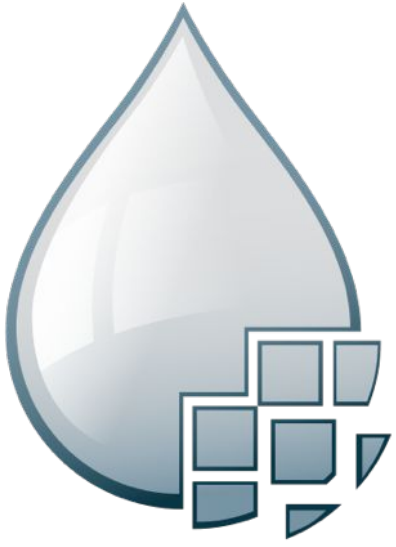
```
$(document).ready(function() {
    var t = $('#example').DataTable();

var wsUri =
"ws://pulsar1:8080/ws/v2/consumer/persistent/public/default/trans
com/tc-reader?subscriptionType=Shared";
websocket = new WebSocket(wsUri);
websocket.onopen = function(evt) {
   console.log('open');
};
websocket.onerror = function(evt) {console.log('ERR', evt)};
```

```
websocket.onmessage = function(evt) {
    var dataPoints = JSON.parse(evt.data);
    if ( dataPoints === undefined || dataPoints == null || dataPoints.payload === undefined || dataPoints.payload
== null ) {
            return;
    }
    if (websocket.readyState === WebSocket.OPEN) {
        websocket.send("{\"messageId\": \"" + dataPoints.messageId + "\"}");
    }
    if (IsJsonString(atob(dataPoints.payload))) {
        var pulsarMessage = JSON.parse(atob(dataPoints.payload));
        if ( pulsarMessage === undefined || pulsarMessage == null ) {
                return;
        }
        t.row.add( [ dataPoints.key, dataPoints.publishTime, pulsarMessage.title,
                    pulsarMessage.latitude, pulsarMessage.longitude, pulsarMessage.pubDate,
                    pulsarMessage.ts] ).draw(true );
      }
};
} );
```

https://medium.com/@tspann/transit-watch-real-time-feeds-d98ff62b3bbb

DataFlows for Data Ingest, Movement and Routing

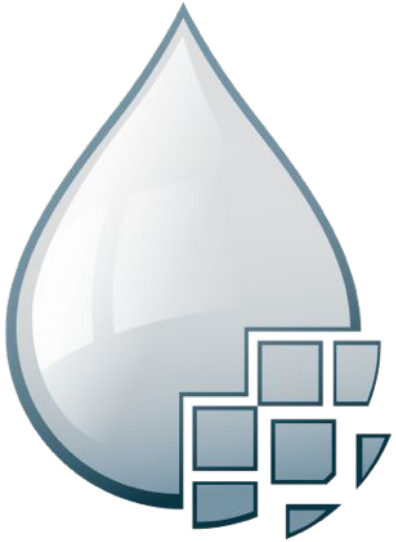


- Guaranteed delivery
- Data buffering
  - Backpressure
  - Pressure release
- Prioritized queuing
- Flow specific QoS
  - Latency vs. throughput
  - Loss tolerance
- Data provenance
- Supports push and pull models

- Hundreds of processors
- Visual command and control
- Over a sixty sources
- Flow templates
- Pluggable/multi-role security
- Designed for extension
- Clustering
- Version Control

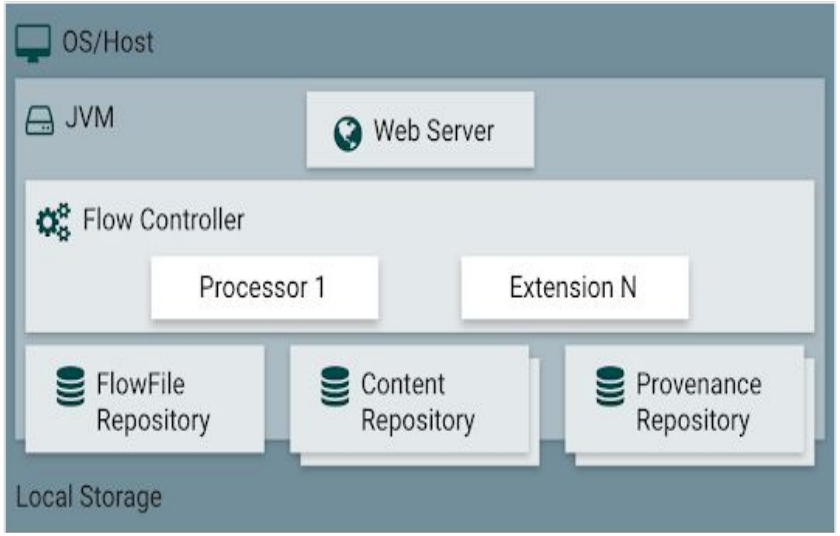
FLaNK
flankstack.dev

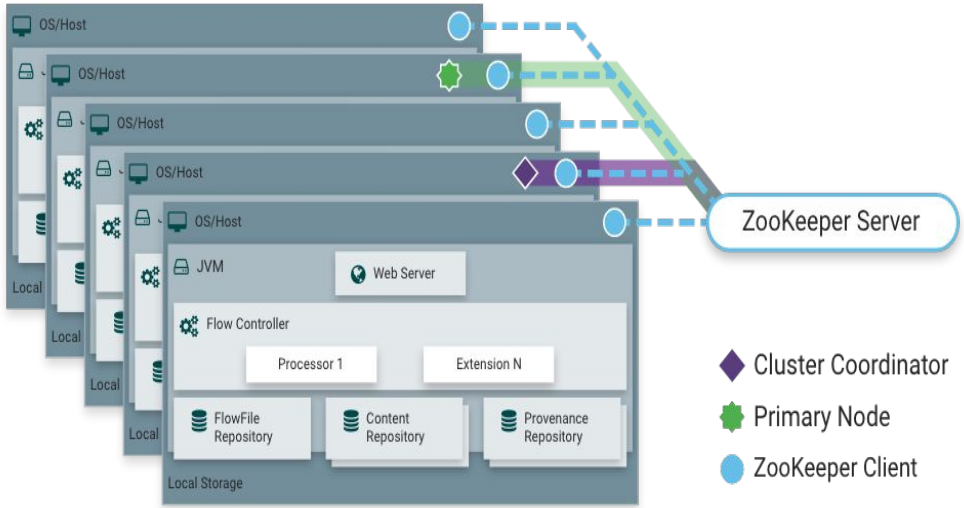
The Power of Apache NiFi



- Moving Binary, Unstructured, Image and Tabular Data
- Enrichment
- Universal Visual Processor
- Simple Event Processor
- Routing
- Feeding data to Central Messaging
- Support for modern protocols
- Kafka Protocol Source/Sink
- Pulsar Protocol Source/Sink

Architecture





https://nifi.apache.org/docs/nifi-docs/html/overview.html

Tim SPANN

https://github.com/tspannhw https://www.datainmotion.dev/

https://www.meetup.com/futureofdata-princeton/