



**Stream  
Native**

# An Introduction to Pulsar's Database Table Abstraction

David Kjerrumgaard | Developer Advocate



**David Kjerrumgaard**  
Developer Advocate

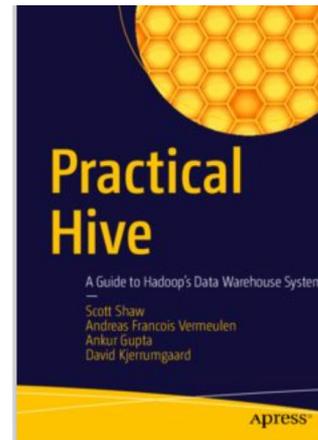
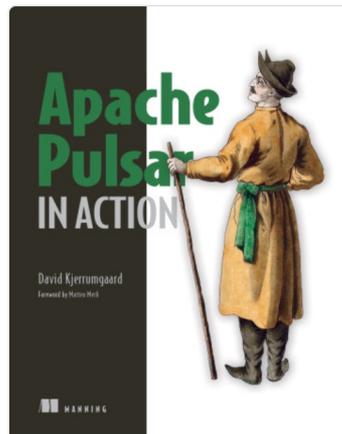
- Apache Pulsar Committer
- Former Principal Software Engineer on Splunk's messaging team that is responsible for Splunk's internal Pulsar-as-a-Service platform.
- Former Director of Solution Architecture at Streamlio.
- Global practice director of Professional Services at Hortonworks.





**David Kjerrumgaard**  
Author

- Author of **Pulsar In Action**.
- Co-Author of *Practical Hive*



<https://streamnative.io/download/manning-ebook-apache-pulsar-in-action>

# TableView

## Pulsar's Database Table Abstraction

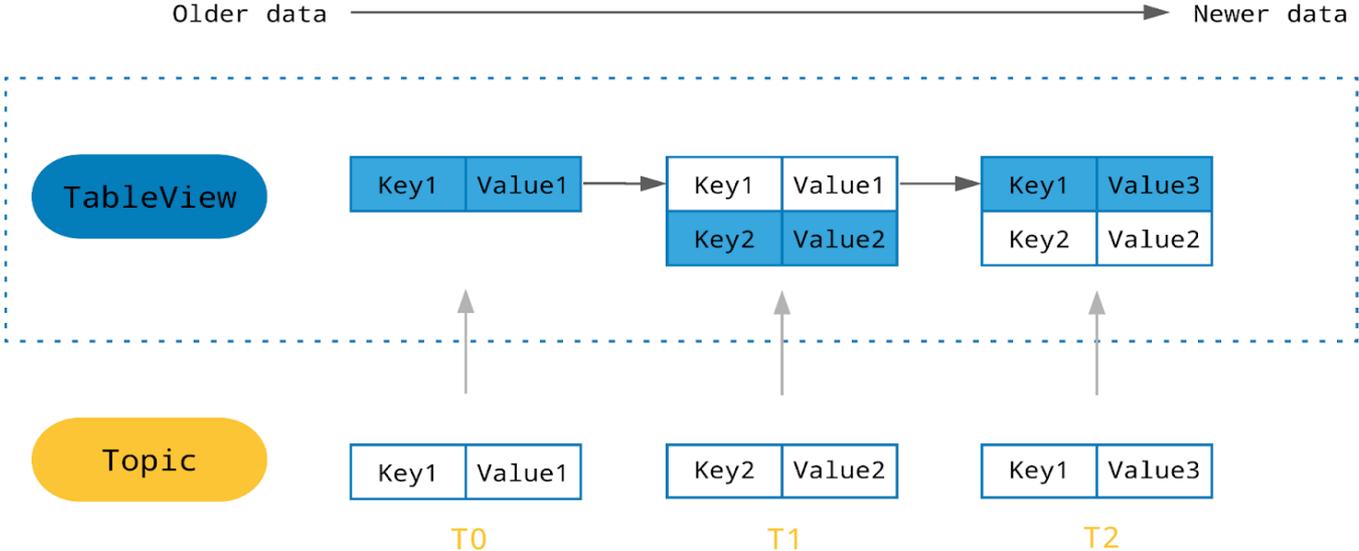
# Why a Table Abstraction?

- In many use cases, applications want to consume data from a Pulsar Topic as if it were a database table, where each new message is an “update” to the table.
- Up until now, applications used Pulsar consumers or readers to fetch all the updates from a topic and construct a map with the latest value of each key for received messages.
- The Table Abstraction provides a standard implementation of this message consumption pattern for any given keyed topic.

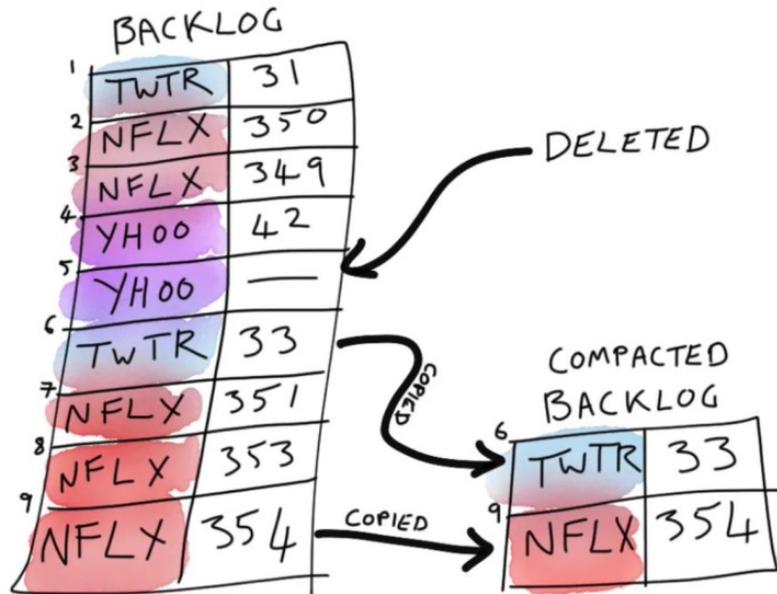
# TableView

- New Consumer type added in Pulsar 2.10 that provides a continuously updated key-value map view of compacted topic data.
- An abstraction of a changelog stream from a primary-keyed table, where each record in the changelog stream is an update on the primary-keyed table with the record key as the primary key.
- READ ONLY DATA STRUCTURE!

# What is it?



# How does it work?



- When you create a TableView, and additional compacted topic is created.
- In a compacted topic, only the most recent value associated with each key is retained.
- A background reader consumes from the compacted topic and updates the map when new messages arrive.

# Working with TableView

# Building a TableView

```
TableView<StockQuote> stockQuoteTable = pulsarClient
    .newTableViewBuilder(Schema.JSON(StockQuote.class))
    .autoUpdatePartitionsInterval(2, TimeUnit.SECONDS)
    .topic("persistent://public/default/stock-quotes")
    .create();
```

# Reading from a TableView

Method	Description
<code>get(String key)</code>	Returns the value to which the specified key is mapped, or null if this map contains no mapping for the key.
<code>forEach(BiConsumer&lt;String,T&gt; action)</code>	Performs the given action for each entry in this map until all entries have been processed or the action throws an exception.
<code>forEachAndListen(BiConsumer&lt;String,T&gt; action)</code>	Performs the given action for each entry in this map and registers a listener which triggers the action to be performed when the table is updated.

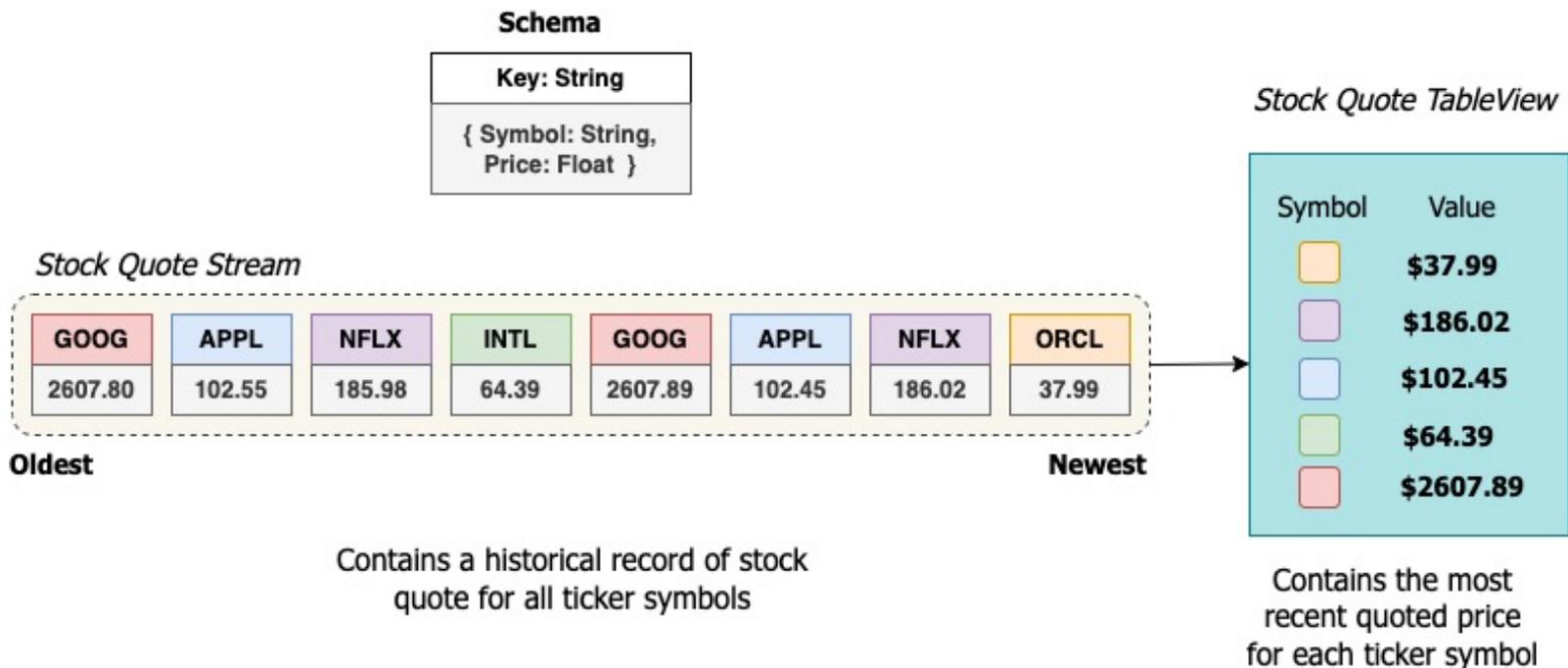
# How to use the API?

- The `get()` method can be used to join a table to an active stream consumer.
- The `forEach()` method can be used to scan the table and perform an action on the contents one time.
- The `forEachAndListen()` method can be used to scan the table and perform an action on the contents periodically.

# Use Case

## Stock Trading Platform

# Stock Quotes Topic



# Stock Trades Topic

## Schema

Key: String
{ Symbol: String, NumShares: Integer, ExecutedPrice: Float }

## Stock Trade Stream

<b>GOOG</b> 125 2607.80	<b>APPL</b> 400 102.55	<b>NFLX</b> 175 185.98	<b>INTL</b> 300 64.39	<b>GOOG</b> 50 2607.89	<b>APPL</b> 500 102.45	<b>NFLX</b> 200 186.02	<b>ORCL</b> 100 37.99
-------------------------------	------------------------------	------------------------------	-----------------------------	------------------------------	------------------------------	------------------------------	-----------------------------

Oldest

Contains a historical record of executed stock trades for all ticker symbols

Newest

## Stock Stats Calculator



## Stock Stats Stream

<b>ORCL</b> 37.99 39.30 320,490
--

# Stock Statistics Topic

## Schema

Key: String
{ Symbol: String, LowPrice: Float, HighPrice: Float, DailyVolume: Long }

Stock Stats  
Calculator



Stock Stats Stream

GOOG	APPL	NFLX	INTL	GOOG	APPL	NFLX	ORCL
2607.89	99.50	175.01	59.90	2607.89	99.50	175.01	37.99
2623.25	102.55	185.98	64.39	2623.25	102.98	191.50	39.30
102,225	567,790	1,186,950	980,120	102,725	567,890	1,187,350	320,490

Oldest

Newest

Changelog of computed  
statistics for all stocks.

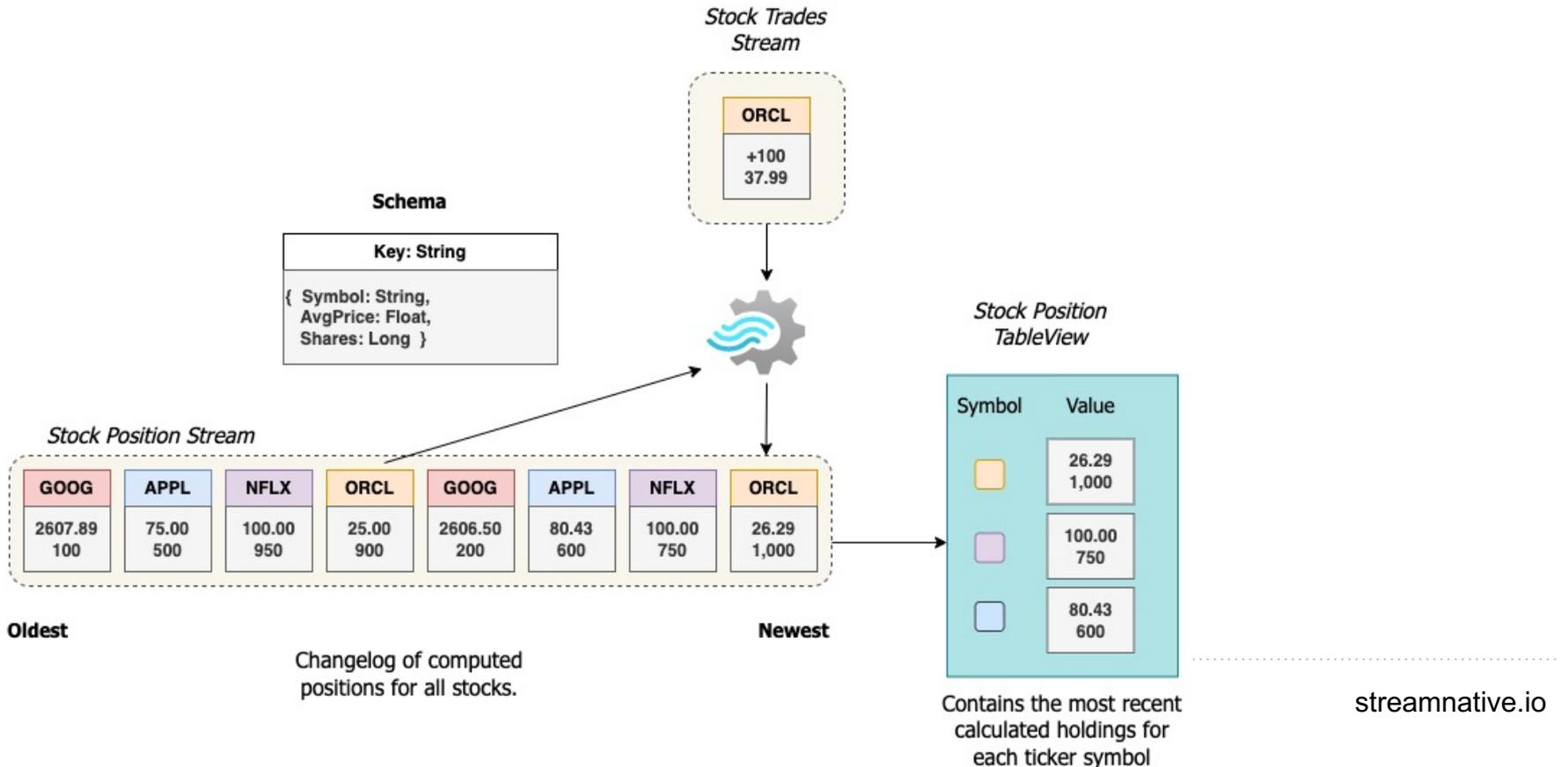
Stock Stats TableView

Symbol	Value
GOOG	37.99 39.30 320,490
NFLX	186.02 191.50 1,187,350

Contains the most recent  
calculated statistics for  
each ticker symbol

streamnative.io

# Stock Positions Topic



# TableView Patterns

- Table Lookup
- Single Scan.
- Periodic Scan.

# Join Streams to TableView

```
Select * from StockQuoteStream Q LEFT JOIN  
StockPositionTable T ON T.symbol = Q.symbol
```

*Stock Quote Stream*

GOOG	APPL	NFLX	INTL	GOOG	APPL	NFLX	ORCL
2607.80	102.55	185.98	64.39	2607.89	102.45	186.02	37.99

```
{Last Price: $37.99,  
Shares: 1,000  
Net Gain: $11,700  
Percent Gain: 44.5%}
```

*Stock Position  
TableView*

Symbol	Value
GOOG	26.29 1,000
NFLX	100.00 750
APPL	80.43 600

```
public void join() throws PulsarClientException {
    table = getClient().newTableViewBuilder(Schema.JSON(StockPosition.class))
        .autoUpdatePartitionsInterval(interval: 2, TimeUnit.SECONDS)
        .topic(tableTopic)
        .create();

    stockQuoteStream = getClient().newConsumer(Schema.JSON(StockQuote.class))
        .subscriptionName("portfolio")
        .topic(streamTopic)
        .messageListener((con, msg) -> {
            StockPosition position = table.get(msg.getValue().getTickerSymbol());
            if (position != null) {
                System.out.println(String.format
                    ("%s [ Last Trade: $%.2f Purchase Price: $%.2f Shares: %,d Net Gain: $%,.2f ]",
                     msg.getValue().getTickerSymbol(), msg.getValue().getQuotePrice(),
                     position.getPurchasePrice(), position.getQuantity(),
                     ((msg.getValue().getQuotePrice() - position.getPurchasePrice()) *
                      position.getQuantity()));
            }
        });
    try {
        con.acknowledge(msg);
    } catch (PulsarClientException e) {
        e.printStackTrace();
    }
}
```

# Join TableView to TableView

*Stock Quote TableView*

Symbol	Value
	<b>\$37.99</b>
	<b>\$186.02</b>
	<b>\$102.45</b>
	<b>\$64.39</b>
	<b>\$2607.89</b>

*Stock Stats TableView*

Symbol	Value			
	<table><tr><td>37.99</td></tr><tr><td>39.30</td></tr><tr><td>320,490</td></tr></table>	37.99	39.30	320,490
37.99				
39.30				
320,490				
	<table><tr><td>186.02</td></tr><tr><td>191.50</td></tr><tr><td>1,187,350</td></tr></table>	186.02	191.50	1,187,350
186.02				
191.50				
1,187,350				



```
{ Symbol: ORCL,  
  Last Price: $37.99,  
  Day Low: $37.99,  
  Day High: $39.30,  
  Daily Volume: 320,49 }
```

```
public void join() throws PulsarClientException {
    stockQuoteTable = getClient().newTableViewBuilder(Schema.JSON(StockQuote.class))
        .autoUpdatePartitionsInterval(interval: 2, TimeUnit.SECONDS)
        .topic(stockQuoteTopic)
        .create();

    stockStatisticsTable = getClient().newTableViewBuilder(Schema.JSON(StockStatistics.class))
        .autoUpdatePartitionsInterval(interval: 2, TimeUnit.SECONDS)
        .topic(stockStatsTopic)
        .create();

    stockQuoteTable.forEach((symbol, quote) -> {
        StockStatistics stats = stockStatisticsTable.get(symbol);

        if (stats != null) {
            System.out.println(String.format("%s [Quote: %.2f Low: %.2f High: %.2f Volume: %,d Time: %s",
                symbol, quote.getQuotePrice(), stats.getLowValue(),
                stats.getHighValue(), stats.getTradeVolume(), stats.getTimestamp()));
        }
    });
}
```

# Continuous TableView Join

(Slow Changing Table)

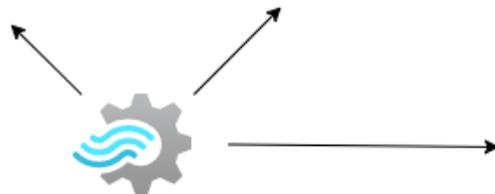
(Fast Changing Table)

*Stock Position TableView*

Symbol	Value
	30.00 1,000
	100.00 750
	80.43 600

*Stock Quote TableView*

Symbol	Value
	<b>\$37.99</b>
	<b>\$186.02</b>
	<b>\$102.45</b>
	<b>\$64.39</b>
	<b>\$2607.89</b>



```
{  
  Last Price: $37.99,  
  Shares: 1,000  
  Net Gain: $7,990  
  Percent Gain: 26.63%  
}
```

```
public void join() throws PulsarClientException {
    getStockPositionTable().forEachAndListen((symbol, position) -> {
        StockQuote quote = null;
        try {
            quote = getStockQuoteTable().get(symbol);
        } catch (PulsarClientException e) {
            e.printStackTrace();
        }

        if (quote != null) {
            System.out.println(String.format
                ("%s [ Last Trade: $%.2f Purchase Price: $%.2f Shares: %,d Net Gain:
                 symbol, quote.getQuotePrice(), position.getPurchasePrice(),
                 position.getQuantity(),
                 ((quote.getQuotePrice()) - position.getPurchasePrice()) * posit
            }
        });
    }
}
```

# DEMO TIME

- Demo 1: Stream/Table Join (Lookup)
- Demo 2: Table/Table Join (Single Scan)
- Demo 3: Table/Table join (Periodic Scan)

<https://github.com/david-streamlio/table-view-demo>

# Summary

- Pulsar's new TableView API provides a database table abstraction on top of Topic data.
- I demonstrated how it can be used to emulate SQL joins with Streams or other TableViews.
  - Join Streams to TableViews
  - Join TableViews to Other TableViews

# Thanks for attending!

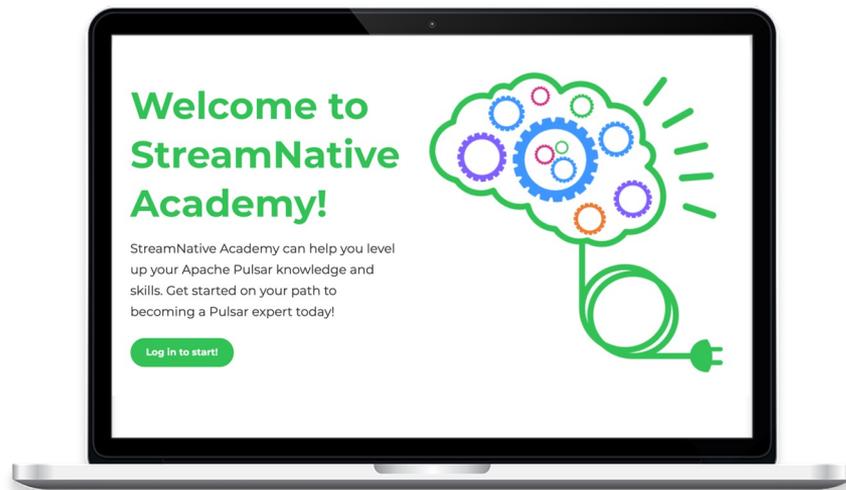
Scan the QR Code to learn more about Apache Pulsar.

## Explore the Code:

<https://github.com/david-streamlio/table-view-demo>

# Now Available On-Demand Pulsar Training

[Academy.StreamNative.io](https://Academy.StreamNative.io)



Platform Engineer [Remote]

San Francisco

Platform Engineer (Flink/Spark) [Remote]

San Francisco

Product Engineer - Cloud [Remote]

San Francisco

Platform Engineer (Flink/Spark) [Remote]

San Francisco

Product Engineer - Cloud [Remote]

San Francisco

Sr. Product Manager [Remote]

San Francisco

# We're Hiring

[streamnative.io/careers/](https://streamnative.io/careers/)

# Questions

# Let's Keep in Touch!



**David Kjerrumgaard**

Developer Advocate



@Dkjerrumg1



<https://www.linkedin.com/davidkj>



<https://github.com/david-streamlio>