

Platformatic

View online

Download PDF

Reimagining Kafka for Node.js

Paolo Insogna
Node.js TSC, Principal Engineer

Let's dive into the unknown!



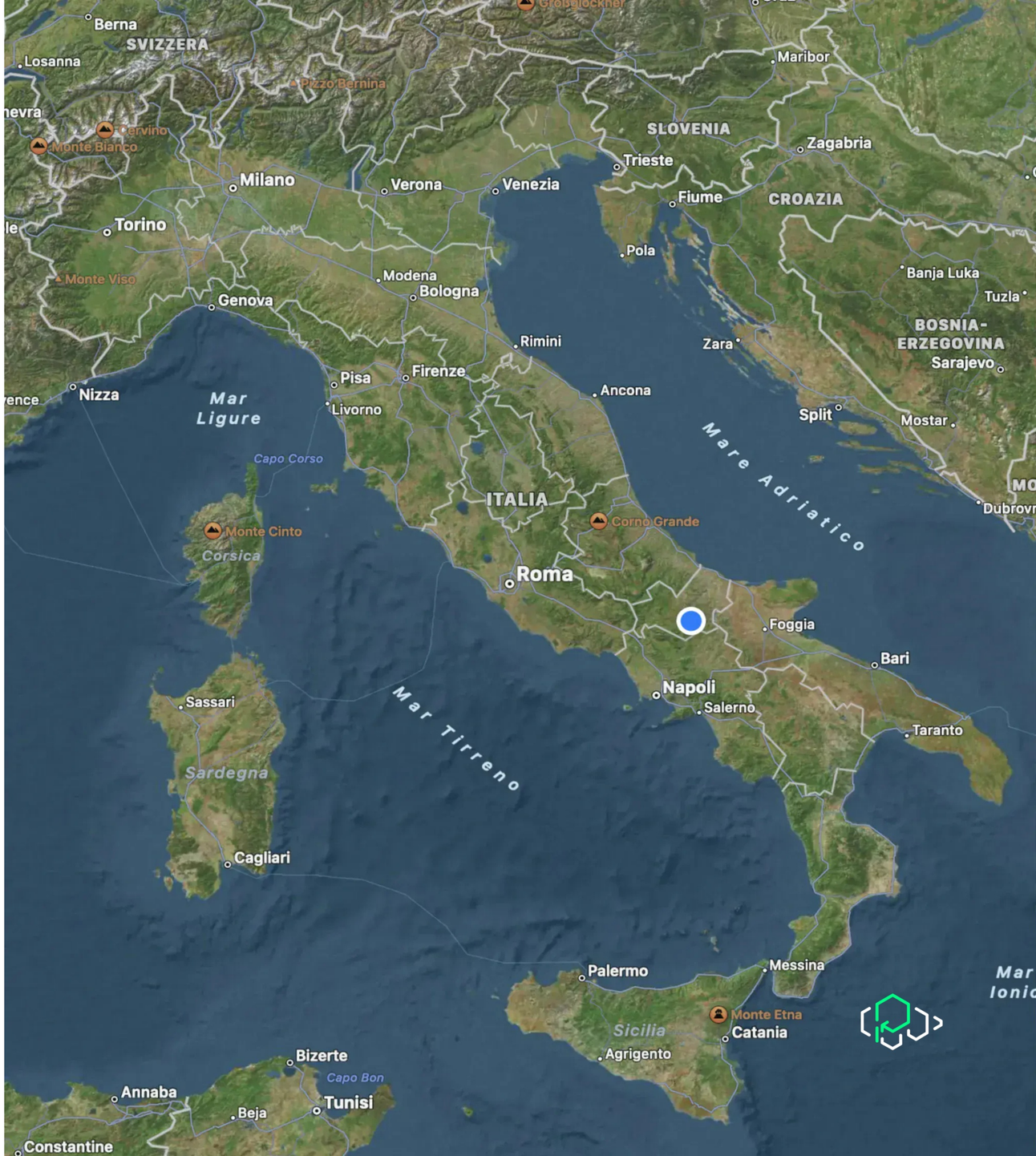
Hello, I'm **Paolo**!



Node.js Technical Steering Committee Member

Platformatic Principal Engineer

`paoloinsogna.dev` `ShogunPanda` `p_insogna` `pinsogna`



Some companies trusting us

Booking.com

Article

supabase

Case Study

SPENDESK

Case Study

Hello Apache Kafka!

A key component of the world

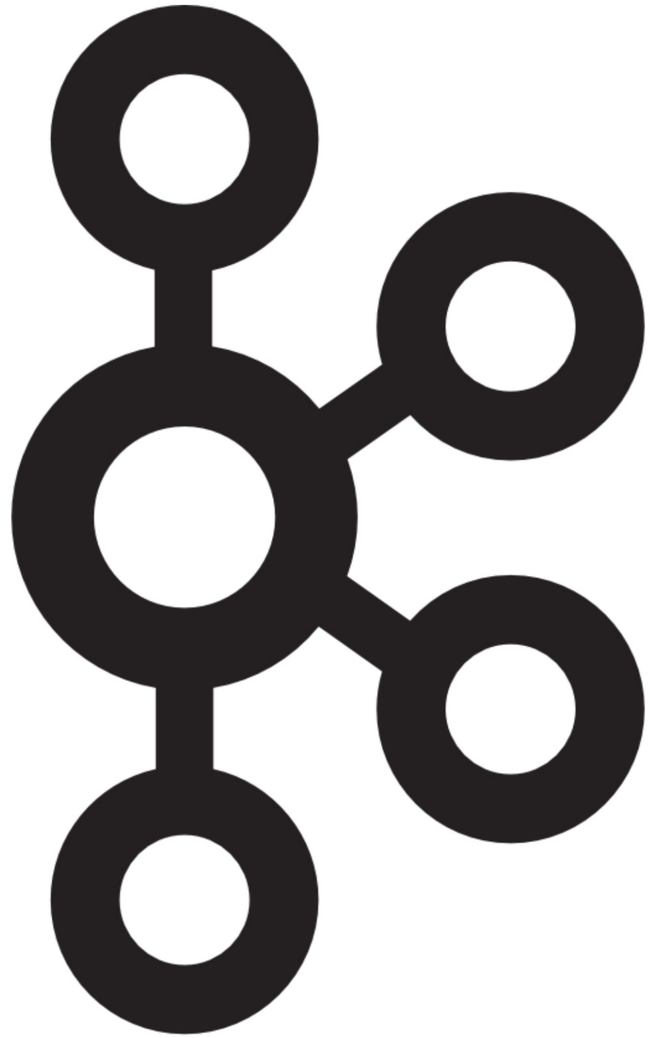
It is undoubtedly the backbone of modern event-driven systems.

Used by the giants

Powers real-time data processing for companies like Netflix, LinkedIn, and Uber.

Impressive performance

Millions of messages with minimal latency.

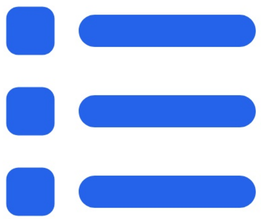


How does it work?



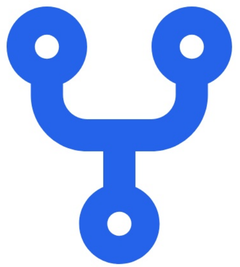
Brokers

Kafka servers that store and serve data across a distributed cluster.



Topics

Named streams of records, like channels for organizing messages.



Partitions

Topics are split into partitions for parallelism and scalability.

How do we use it?



Producers

Applications that write/publish messages to Kafka topics.



Consumers

Applications that read/subscribe to messages from topics.



Consumer Groups

Consumers are organized into groups for load balancing.

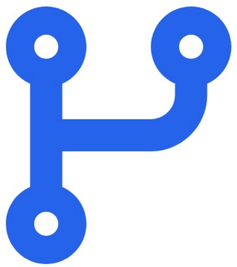


Partition Assignment

Each partition is assigned to **only one** consumer within a group.



Kafka API Versioning



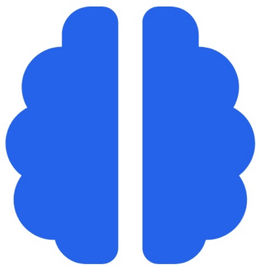
Dynamic versioning

Evolution with backward compatibility considerations.



Independent advancement

Each API has its own independent version number.

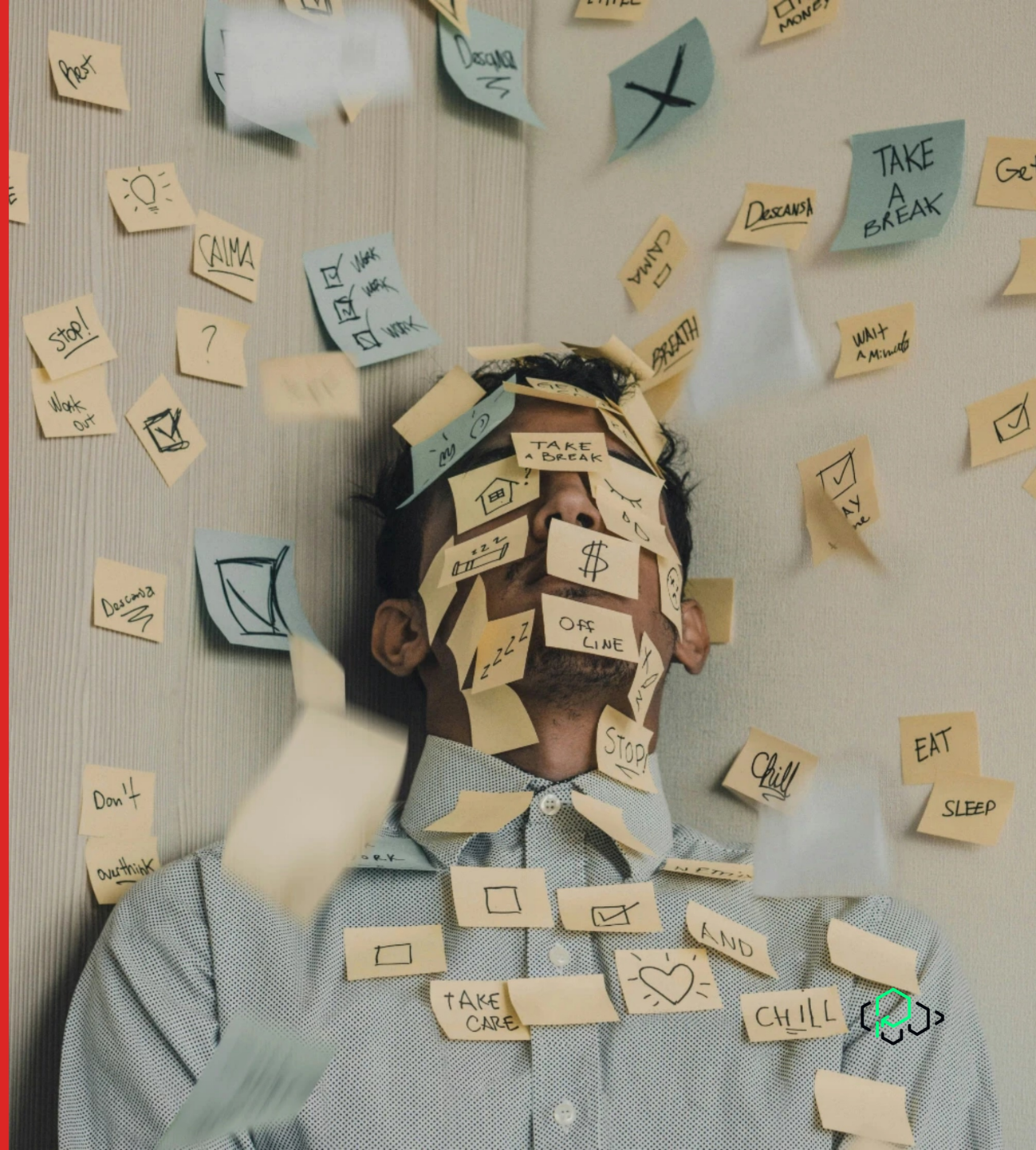


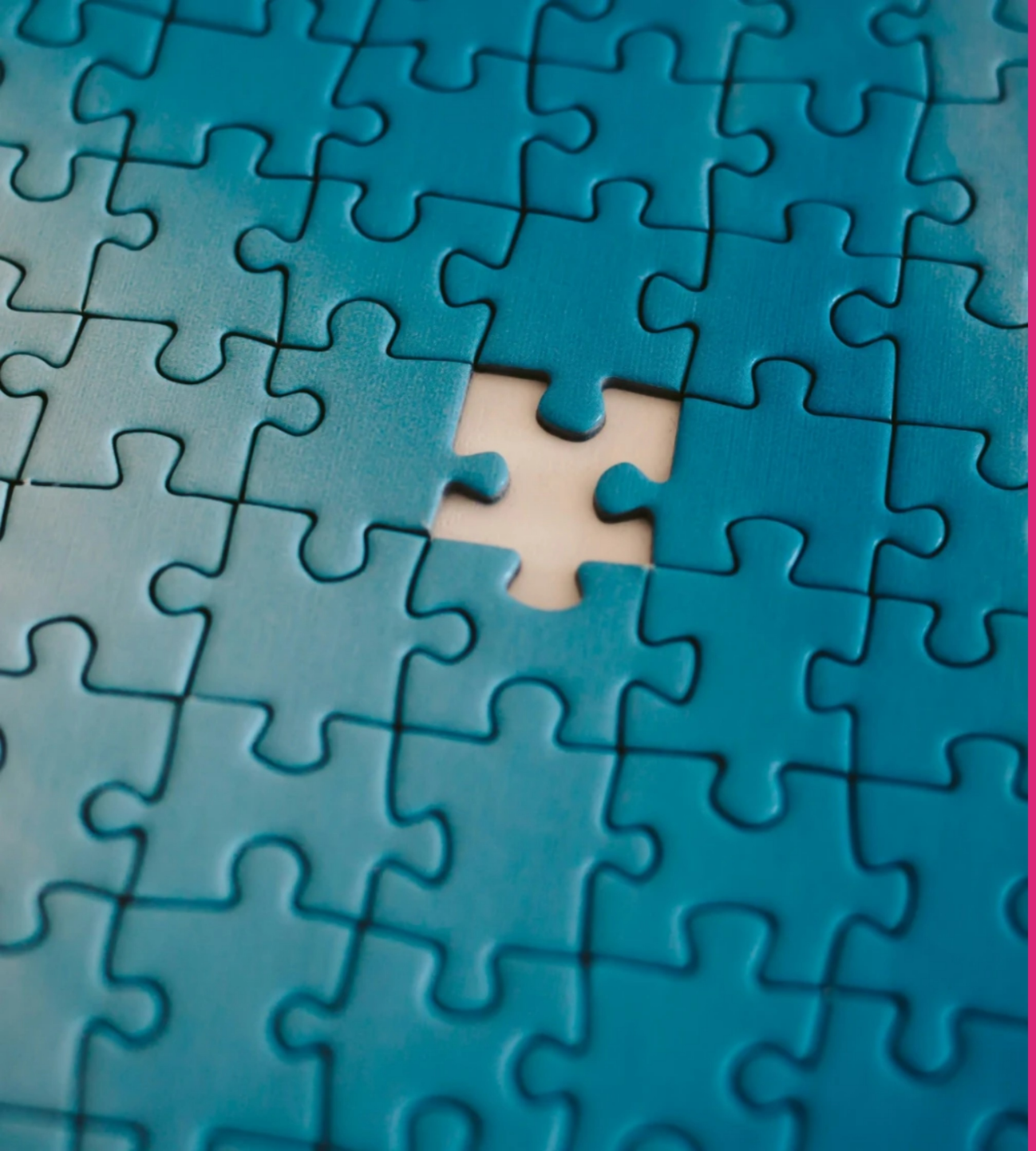
Protocol complexity

Clients must handle multiple API versions and features.

That's a lot to handle ...

... especially if, like me,
you **have never** used Kafka! 😂





...and it got much worse! 😭

Where is your documentation?



No client building support

Low-level API documentation is scattered and incomplete.



KIP archaeology

You need to dig through Kafka Improvement Proposals (KIP) to understand APIs.



Implementation details hidden

Critical protocol details buried in JIRA tickets and mailing lists.

What about Node.js?



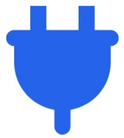
There were already some solutions

node-rdkafka
https://github.com/Blizzard/node-rdkafka

KafkaJS
https://kafka.js.org/



node-rdkafka



Native dependency

Based on native C library (librdkafka).



Compatibility issues

No Worker Threads support.



Cumbersome API

The API is clunky and hard to use.

```
import RDKafka from 'node-rdkafka'

const producer = new RDKafka.Producer(
  {
    'client.id': 'id',
    'metadata.broker.list': 'localhost:9092',
    dr_cb: true
  }
)

producer.on('delivery-report', () => {
  producer.disconnect()
})

producer.connect({}, () => {
  producer.setPollInterval(1)

  producer.produce(
    'topic', 0,
    Buffer.from('value'), 'key',
    -1, null,
    [{ a: '123', b: '456' }]
  )
})
```



KafkaJS



Abandoned

It has not been maintained in the last years.



Latest Kafka features missing

Incomplete protocol support due to inactivity.



Complex and unperformant API

Caused to unnecessary memory allocation.

```
import { Kafka } from 'kafkajs'

const client = new Kafka({
  clientId: 'id', brokers: ['localhost:9092']
})

const consumer = client.consumer({
  groupId: 'group'
})

await consumer.connect()
await consumer.subscribe({ topics: ['topic'] })

await consumer.run({
  async eachMessage ({ message}) {
    console.log('Received message', message)
    return consumer.disconnect()
  }
})
```



We needed a better solution ...

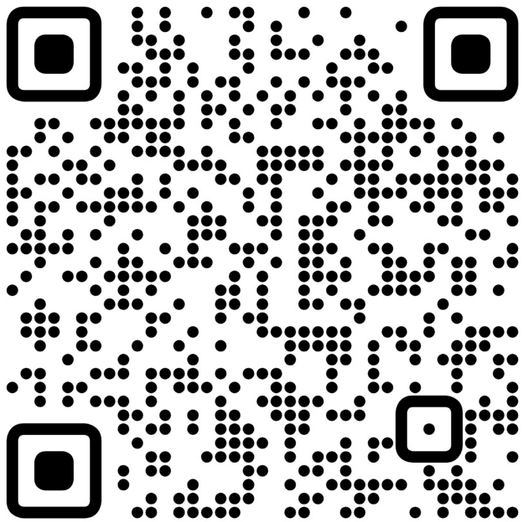


… so we
started fresh!



Hello @platformatic/kafka!

A modern, easy to use and high-performance Kafka client for Node.js



https://github.com/platformatic/kafka

How did we build a client with no documentation available?



AI to the Rescue



Documentation synthesis

AI synthesized scattered information into coherent implementation guides.



Hallucinations were common

So the information had to be fact-checked. But it was still faster than manual research.



Life Saver

Without AI assistance, this project would have taken months longer.

Design Principles



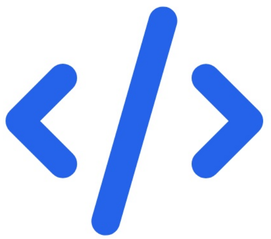
The developer is the center

We prioritized developer experience above all.



Performance matters

We avoided unnecessary memory allocations and copies.

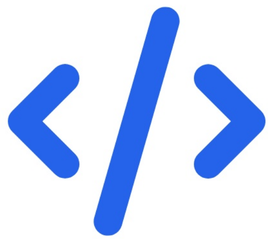


TypeScript first

Our first Typescript experiment. Big thanks to **type stripping**.

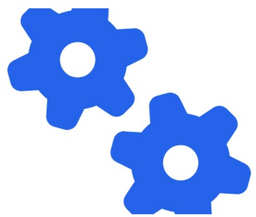


Integrated Serialization and Deserialization



Type-safe

The package is typed end-to-end.
Your IDE will love it.



Cached (de)serializers

This leads to massive V8
optimizations.



Zero copying overhead

Avoids unnecessary data copying
during (de)serialization process.

One consuming semantic: Node.js Streams



Stream-only architecture

Built **exclusively** around Node.js streams for optimal performance.



Multiple consumption patterns

Supports callback, promise-based, and `AsyncIterable` approaches.



Natural fit

Perfect integration with Node.js ecosystem.



Benefits of Node.js Streams



Memory efficiency

Process large datasets without loading everything into memory.



Backpressure handling

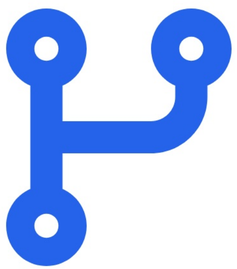
Automatic flow control prevents overwhelming downstream processes.



Error propagation

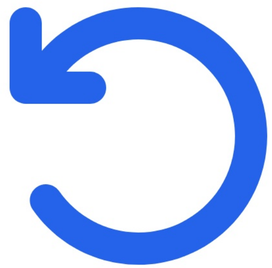
Built-in error handling and graceful failure management.

Every medal has two faces!



Dual API on the outside ...

Supports both callbacks and promises externally for flexibility.



... only callbacks on the inside!

Not a single promise is used internally.

Are we in 2015 again? 🤦


SHUT UP AND
Show the code!
TAKE MY MONEY

Producer API

```
import { Producer, ProduceAcks, stringSerializers } from '@platformatic/kafka'

const producer = new Producer({
  clientId: 'id',
  bootstrapBrokers: 'localhost:9092',
  serializers: stringSerializers
})

await producer.send({
  messages: [
    { key: 'key', value: 'value', headers: { a: '123', b: '456' } }], acks: 0 })
  ],
  acks: ProduceAcks.NO_RESPONSE
})

console.log('The message has been delivered')
await producer.close()
```



Consumer API

```
import { Consumer, stringDeserializers } from '@platformatic/kafka'

const consumer = new Consumer({
  clientId: 'id', group: 'group',
  bootstrapBrokers: 'localhost:9092', deserializers: stringDeserializers
})

const stream = await consumer.consume({ topics: ['topic'] })

// Since streams are AsyncIterable, both following approaches are possible
stream.on('data', message => {
  console.log('Received message', message)
  stream.close(true)
})

// The stream will be implicitly destroyed after exiting the loop
for await (const message of stream) {
  console.log('Received message', message)
  break
}
await consumer.close()
```



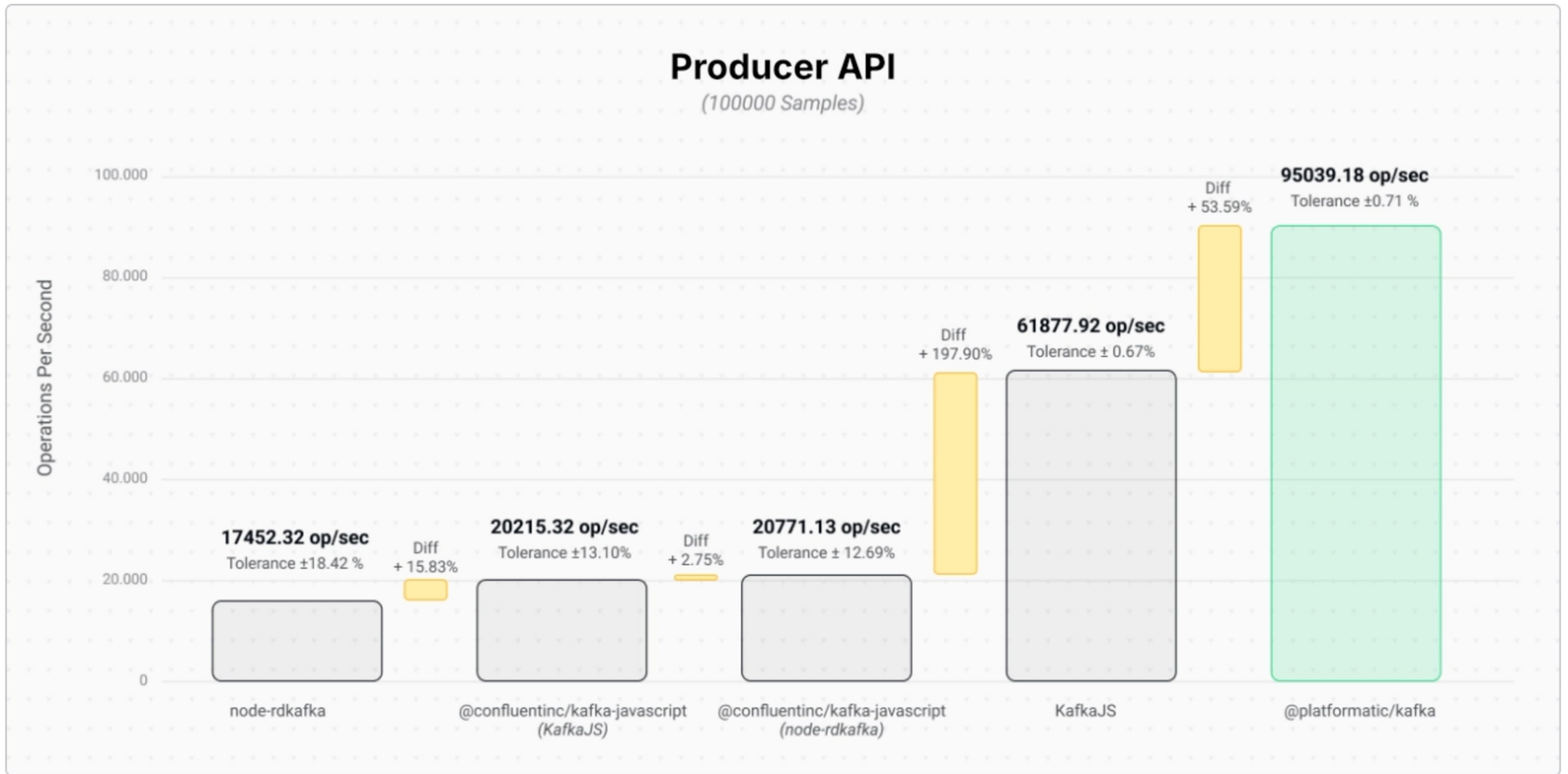
What about performance?



Producer API

Slower tests	Samples	Result	Tolerance	Difference with previous
node-rdkafka	100000	17452.32 op/sec	± 18.42 %	
@confluentinc/kafka-javascript (KafkaJS)	100000	20215.32 op/sec	± 13.10 %	+ 15.83 %
@confluentinc/kafka-javascript (node-rdkafka)	100000	20771.13 op/sec	± 12.69 %	+ 2.75 %
KafkaJS	100000	61877.92 op/sec	± 0.67 %	+ 197.90 %
Fastest test	**Samples**	**Result**	**Tolerance**	**Difference with previous**
@platformatic/kafka	100000	95039.18 op/sec	± 0.71 %	+ 53.59 %

Producer API



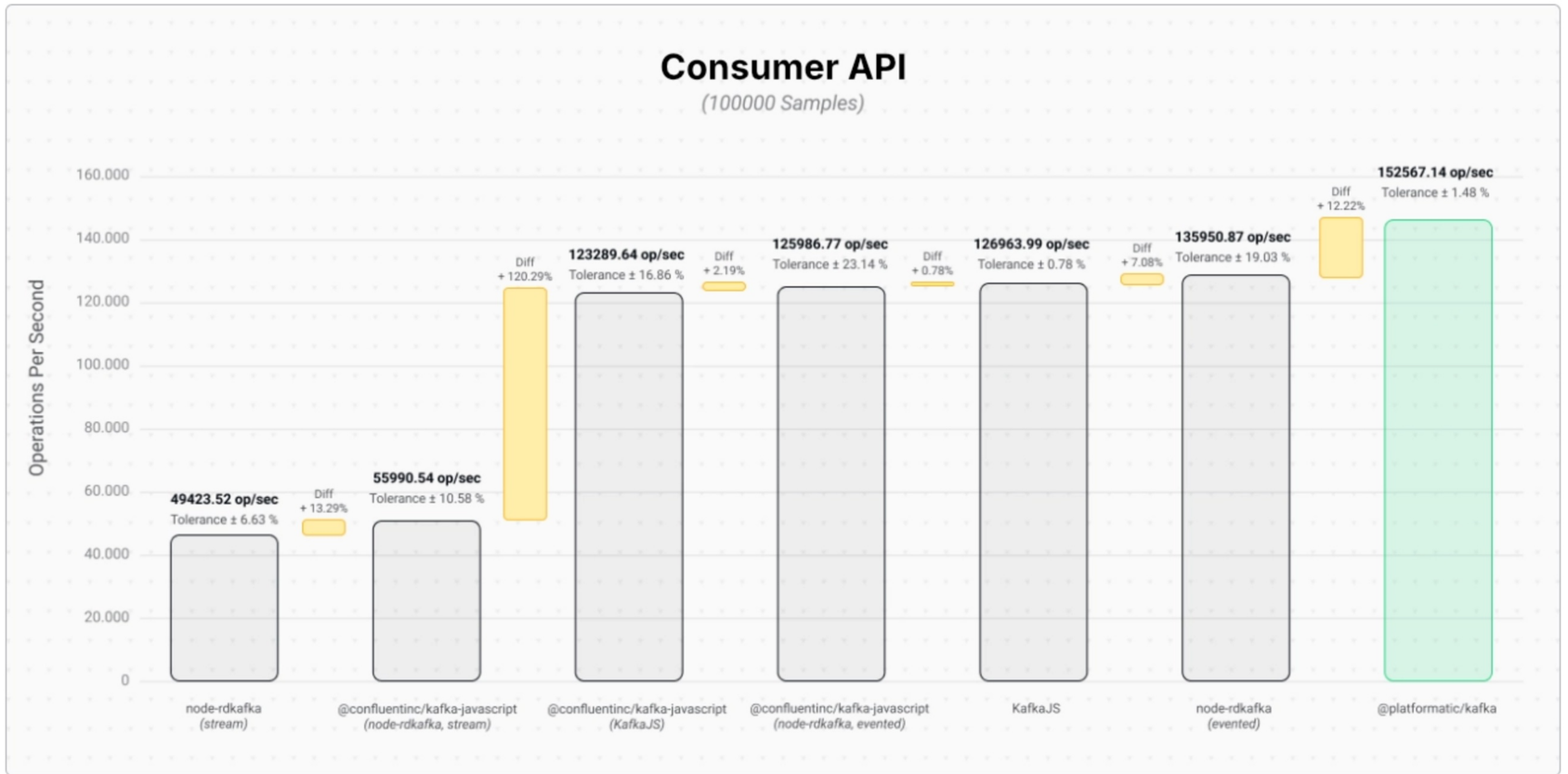
Consumer API

Slower tests	Samples	Result	Tolerance	Difference with previous
node-rdkafka (stream)	100000	49423.52 op/sec	± 6.63 %	
@confluentinc/kafka-javascript (stream)	100000	55990.54 op/sec	± 10.58 %	+ 13.29 %
@confluentinc/kafka-javascript (KafkaJS)	100000	123289.64 op/sec	± 16.86 %	+ 120.20 %
@confluentinc/kafka-javascript (evented)	100000	125986.77 op/sec	± 23.14 %	+ 2.19 %
KafkaJS	100000	126963.99 op/sec	± 4.15 %	+ 0.78 %
node-rdkafka (evented)	100000	135950.87 op/sec	± 19.03 %	+ 7.08 %

Fastest test	Samples	Result	Tolerance	Difference with previous
@platformatic/kafka	100015	152567.14 op/sec	± 1.48 %	+ 12.22 %



Consumer API



One last thing™

“Paths are made by walking.”

Franz Kafka





Thank you!

Paolo Insogna

Node.js TSC, Principal Engineer

@p_insogna

paolo.insogna@platformatic.dev

Platformatic