

[Open in app](#)

# The One Issue Developers Were Not Prepared For With Server-Side Events

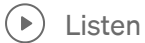
Delve into the lesser-known realm of server-side events and the unexpected issue with our system.



Alexandre Olive · [Follow](#)

Published in ITNEXT

6 min read · Dec 2, 2023



Listen



Share

... More



Photo by [AbsolutVision](#) on [Unsplash](#)

Everything is more complex in a distributed architecture — we should have seen it coming.

We just finished developing our brand-new functionality to generate subtitles for our clients' videos using OpenAI's technology. The architecture is impressive, with microservices, event queues, and workers— enough to make any developer proud. The only thing we were not proud of was using a standard polling system in the browser every 2 seconds to check if the subtitles were ready.

It works, but it could be more efficient, and there are much better solutions.

That's when server-side events or web sockets come into play.

They both allow the connection between the browser and the server to be kept active so that data can flow without making other queries — excellent! They both work perfectly well with one browser and server instance. Still, as you want your application to scale, you'll probably have a load balancer with a unique endpoint and multiple backend instances behind it.

That's where issues start to arise. The connection from the browser to your backend is not on the load balancer; it's on one of the server instances.

Suppose your user connects with SSE and makes a call that needs an asynchronous process, such as sending an event in a queue with a worker.

Once the worker finishes, it needs to notify the precise server instance connected to the user's browser; how do you target the correct one when you can only call the load balancer?

In today's article, I want to use my direct experience as a tech lead working with server-side events on a highly distributed system to share the issues we faced and the solutions we used to overcome them.

## **What are server-side events, and how do they work**

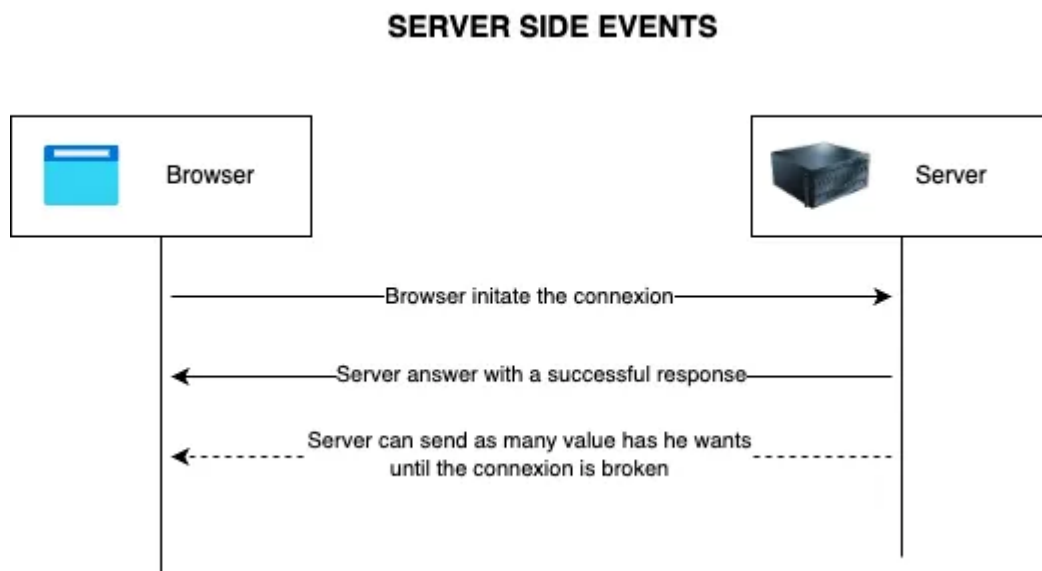
Server Side Events are lesser known browser functionality than their more successful cousin: Websockets.

They both use the same technologies to allow the browser to keep a connection between the browser and the server.

Instead of making an HTTP call that gets an answer directly and closes the connection, it keeps the connection alive even after receiving a response from the server so that the browser and server can keep communicating.

The differences between WebSockets and Server Side Events are:

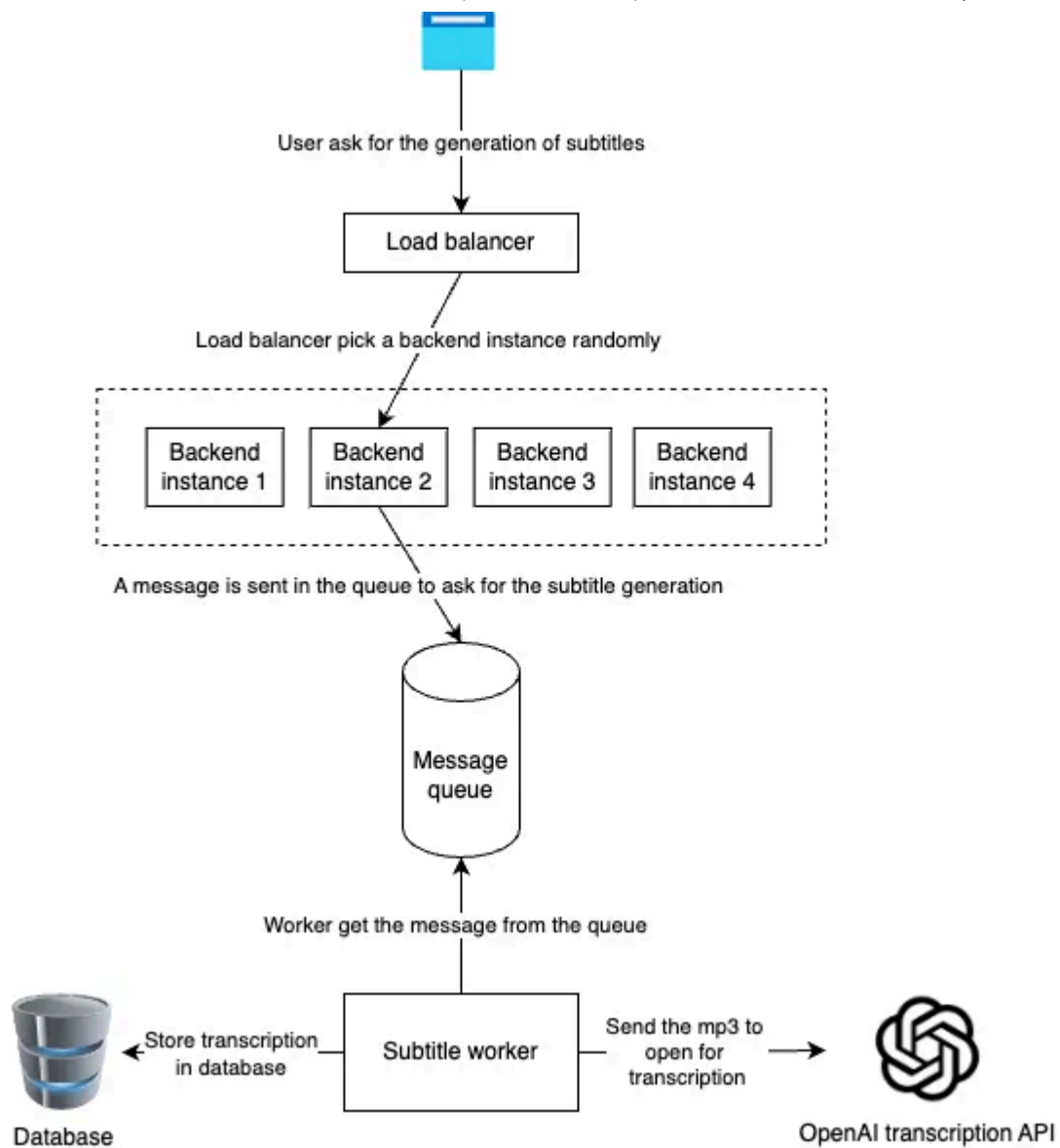
- For Websockets, the exchange is bi-directional; both the browser and server can send events to each other. It's great for things like an interactive chat.
- Server-side events are unidirectional, where only the server can send events to the browser. It's optimal, for example, for a loader to update a percentage of an ongoing action.



Small sequence diagram showing the interaction between browser and server.

The technology and how to implement it are relatively simple — so we thought.

## The simplified distributed architecture

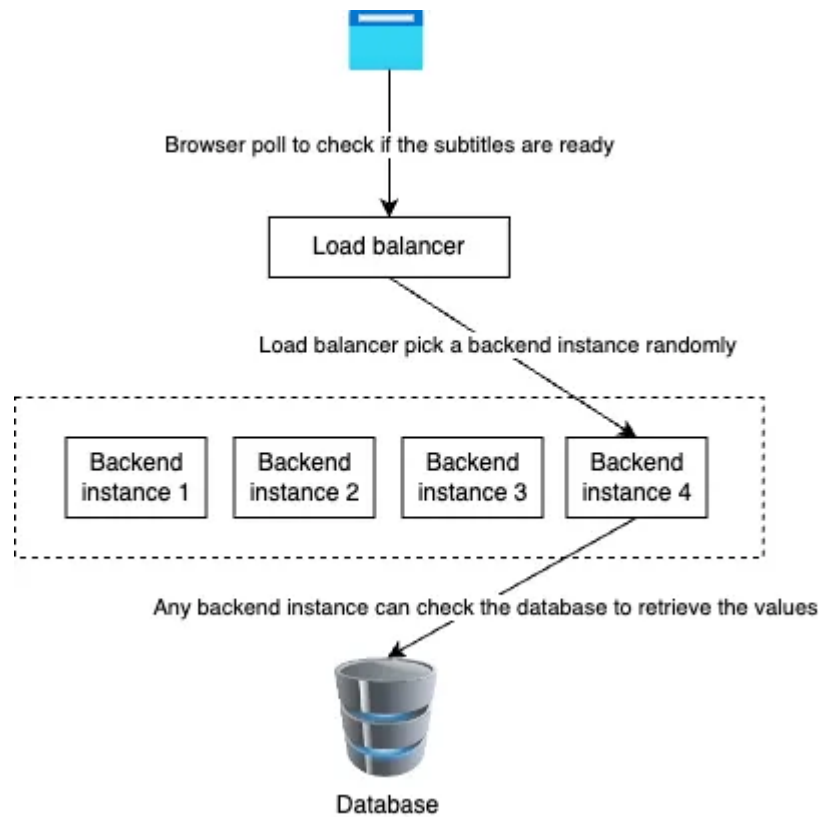


Architecture schema representing a simplified version of our system for subtitle generation

In this schema, I simplified our architecture to remove the noise. We can see what happens when one of our users asks for new subtitles for one of his videos.

- The browser only knows the endpoint for the load balancer, so that's what it calls.
- The load balancer randomly picks one of the backend instances, and the backend then sends a message in the subtitle generation queue.
- The subtitle worker, who is listening to new messages, starts processing the video and uses OpenAI transcription to get the text of the video and then store the content in the database.

I have yet to talk about server-side events on purpose. For the easy polling system, this would be enough.



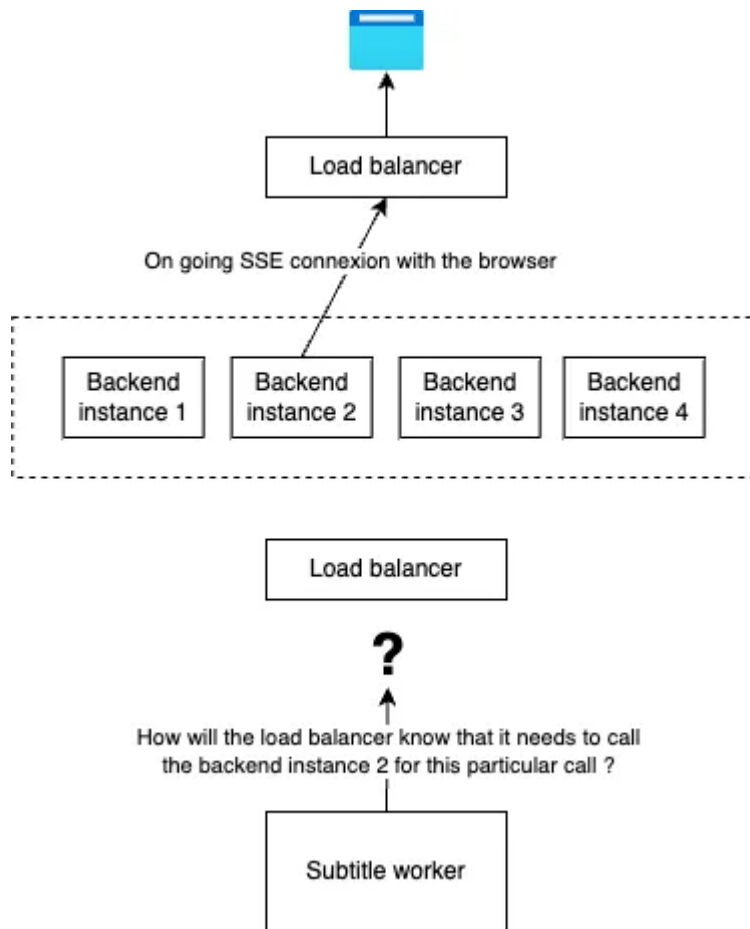
Schema representing the polling system from the browser

Periodically, the browser could send an HTTP call to check if the subtitles are ready; any backend instance could be called to check the database and answer — the end of the story.

But that's not fun; we don't want to poll. We want the server to notify the browser when it's ready.

So, if we look at the first schema again, after the whole process of generating subtitles is over (this schema starts at the bottom).





Schema showing the difficulties of knowing which server instance to call

The SSE connection is between a precise backend instance and the browser. How will the load balancer know that for the call from the subtitle worker, it needs to select instance two and not another one?

## Our thought process and the solution.

We had many ideas on how to fix this issue — most of them were terrible.

Our issue is: “How do we notify the correct backend instance that the process the user is waiting for is finished.”

The first solution we thought of implementing was storing the IP address of the server instance linked to the subtitle ID in an external cache. So that when the worker has finished generating the subtitle, we can get the list of IPs (if more than one user is waiting for the result) and notify all the instances that it's finished.

On paper, this could work, but there are some issues.

The server instances are auto-scaling up and down; the orchestrator could kill a server at any moment, even with an ongoing connection. Server-side events try to reconnect automatically when the connection is lost so that it can reattach to another server without issues.

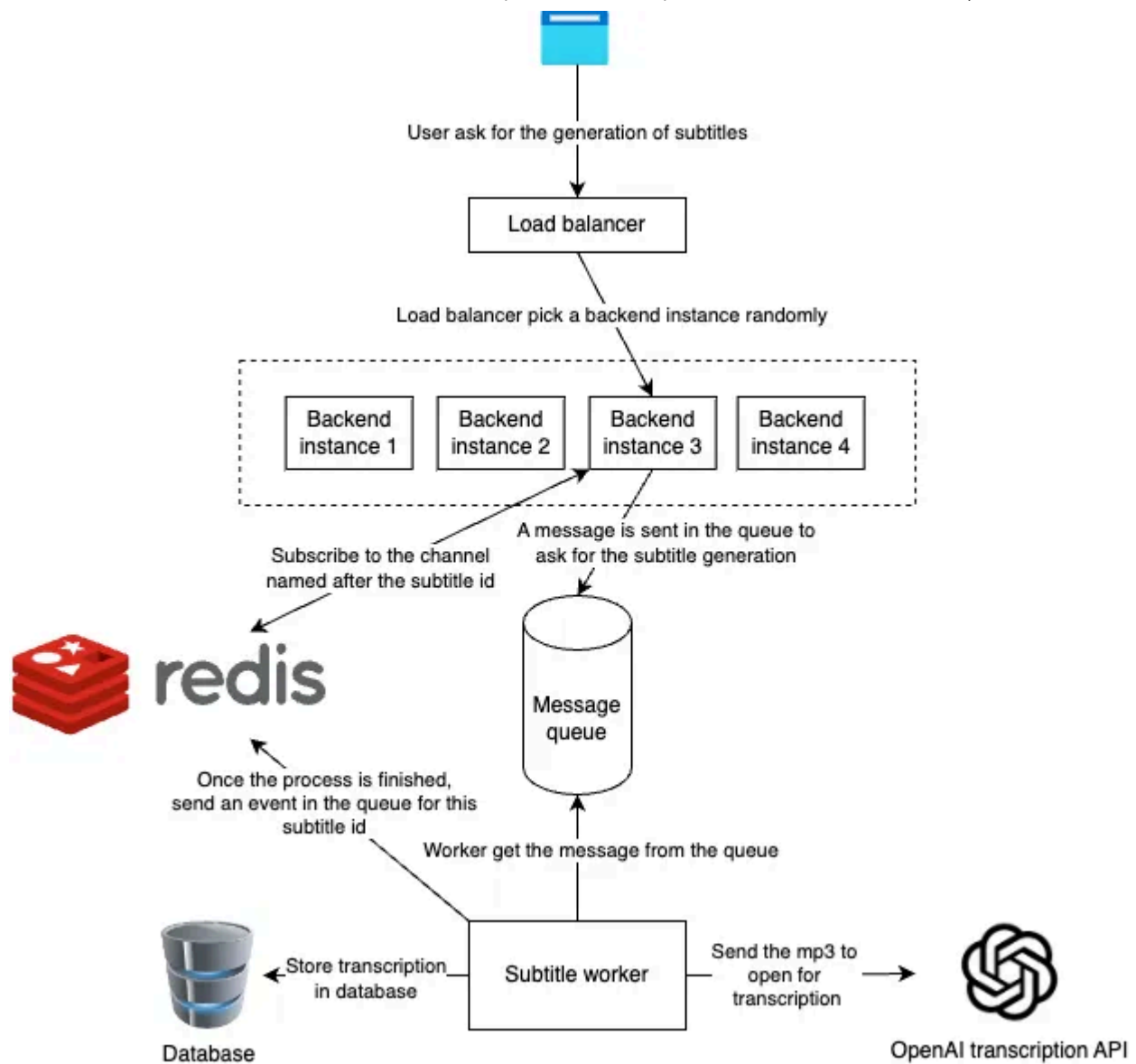
We would still need to introduce a way to clean the cache to remove the server's IP, which is now gone. Also, it would get lost if the process finishes at this exact moment and tries to call the old IP before the new one is stored.

All in all, there must be better solutions.

### **Our working solution**

Events! We have had the solution in front of our eyes since the beginning of this issue started.

Once the subtitle worker completes the processing, he must send an event to a particular queue. We decided to go with Redis Pub/Sub out of convenience because it was already available and working in our system, but it could be implemented with other queue services.



Full schema of the architecture with redis pub/sub in the middle

When the user asks to create a subtitle for his videos, we create a new queue (channel) for this subtitle ID and subscribe to it. Only the server instance where the SSE connection is open is subscribed to the channel.

If another user lands on the page while the process is ongoing and registers to the server-side event in another instance. This instance will also subscribe to the existing channel.

Once the subtitle worker is done processing the subtitle, all it has to do is send an event to the Redis channel for this subtitle ID. All the server instances registered to it will be able to notify the users still connected to server-side events.

Suppose one of the servers is killed because of the automatic downscale. In that case, the browser will automatically reconnect, and upon reconnection, the server will subscribe to the channel if it's not already.



What was such an easy task initially ended up with us tunnel vision on trying to make a direct call between our subtitle worker and the primary backend, which almost made us give up on server-side events.

We just needed to step back and realize that events were the answer from the beginning — they're everywhere in our app, and it was too big for us to see.

If you're interested in complex architectures, read more about ours with my article about Kubernetes and RabbitMQ below.

### Video Streaming at Scale with Kubernetes and RabbitMQ

Deep dive into the problems video streaming sites face and how they can architect their infrastructure to manage the...

techblog.skeepers.io

*Thank you for reading this article until the end. If you liked this article, please don't hesitate to follow me on [X \(Twitter\)](#) or add me on [Linkedin](#).*

Programming

Distributed Systems

Software Development

Server Side Events

Load Balancing



Follow

**Written by Alexandre Olive**

581 Followers · Writer for ITNEXT

## More from Alexandre Olive and ITNEXT



## 6 Simple Rules To Lead a Better Daily Scrum Meeting

6 min read · Nov 9, 2023



192



7





# Upgrading hundreds of Kubernetes clusters

With automated tests and custom tooling.



with

**Pierre Mavro**

CTO & co-founder @ Qovery



Gulcan Topcu in ITNEXT

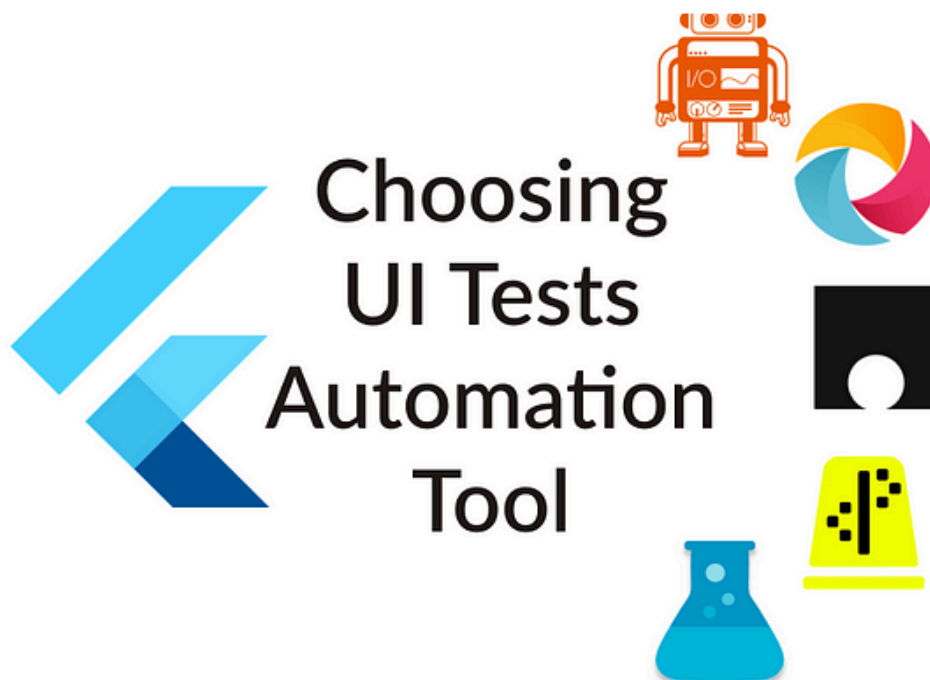
## Upgrading Hundreds of Kubernetes Clusters

Automating the upgrade process for hundreds of Kubernetes clusters is a formidable task, but it's one that Pierre Mavro, the co-founder and...

11 min read · May 6, 2024



110



Pavel Sulimau in ITNEXT

## Flutter: Choosing Mobile UI Tests Automation Tool



In the large-scale Flutter projects our team has been working on for the last 4+ years the topic of UI Tests Automation was always on the...

4 min read · May 5, 2024



252



3



Alexandre Olive

## Transactional Outbox: A Place Where Microservices Architecture And Post-Office Meets

The analogy might be far-fetched, but I'll run with it.

7 min read · Feb 17, 2024



26



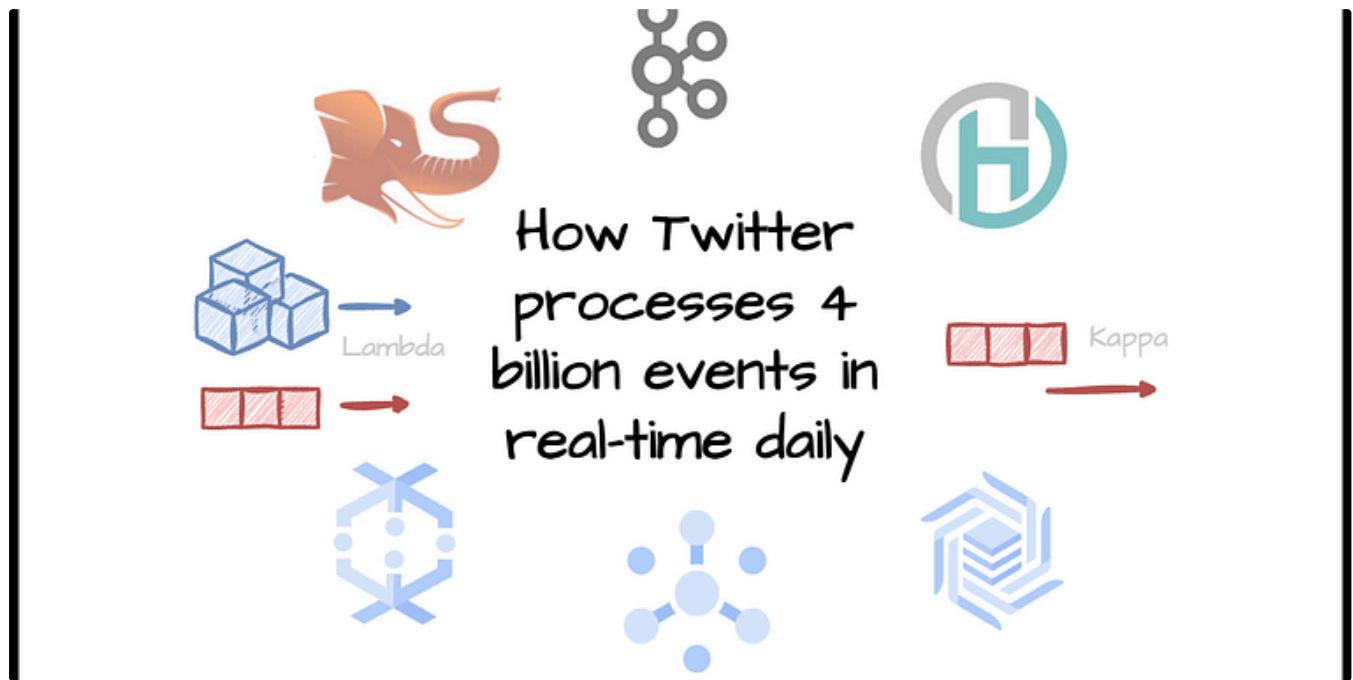
1



See all from Alexandre Olive

See all from ITNEXT

## Recommended from Medium



Vu Trinh in Data Engineer Things

### How Twitter processes 4 billion events in real-time daily

From Lambda to Kappa

6 min read · May 25, 2024

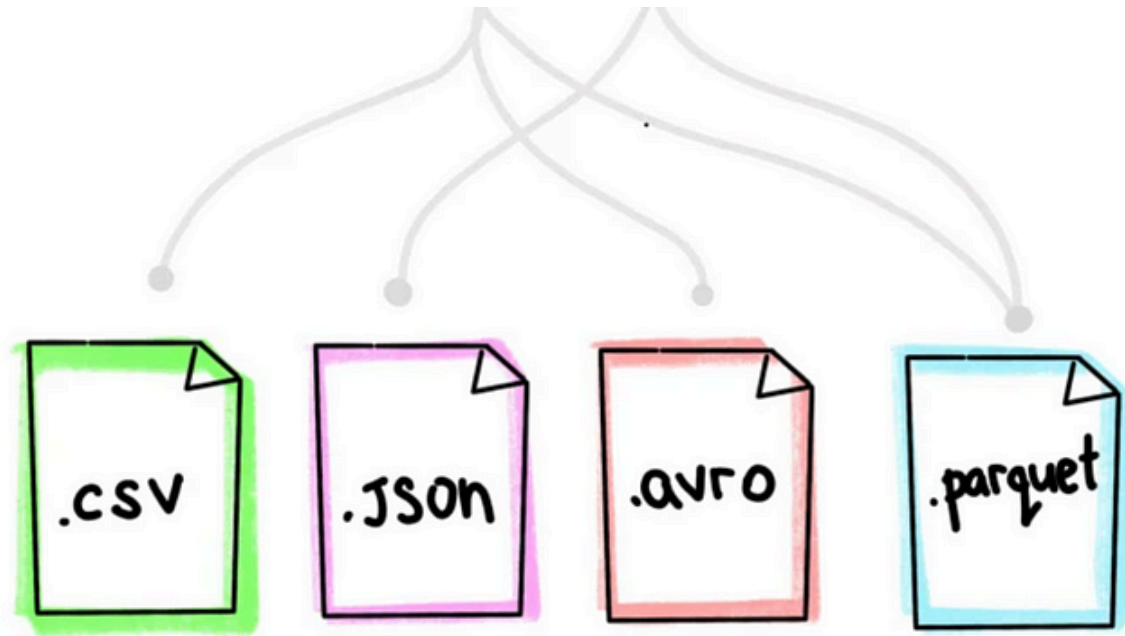



236



1





 Abhinav Vinci

## Why data format matters ? Parquet vs Protobuf vs JSON

Whats data format ?

3 min read · Dec 29, 2023



185



2



### Lists



#### General Coding Knowledge

20 stories · 1270 saves



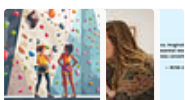
#### Coding & Development

11 stories · 639 saves



#### Stories to Help You Grow as a Software Developer

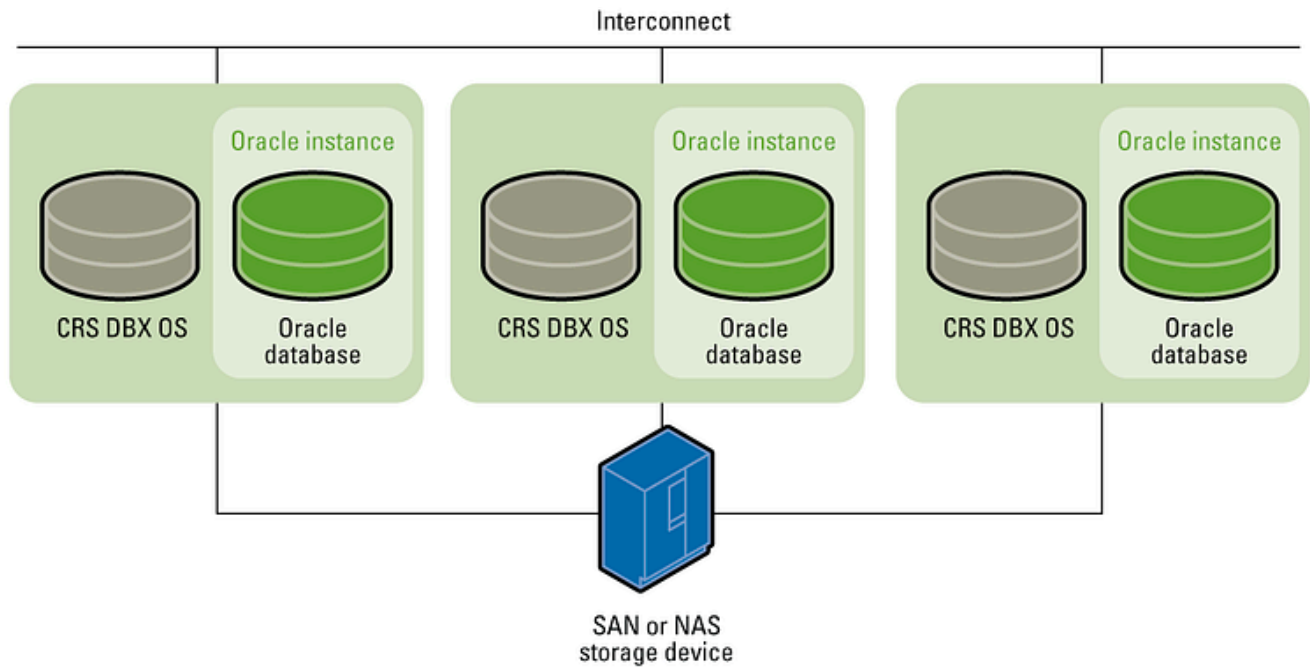
19 stories · 1100 saves




#### Leadership

50 stories · 342 saves





 Devansh in CodeX

## How Pinterest Scaled to 11 Million Users With Only 6 Engineers

Scaling Pinterest — From 0 to 10s of Billions of Page Views a Month in Two Years

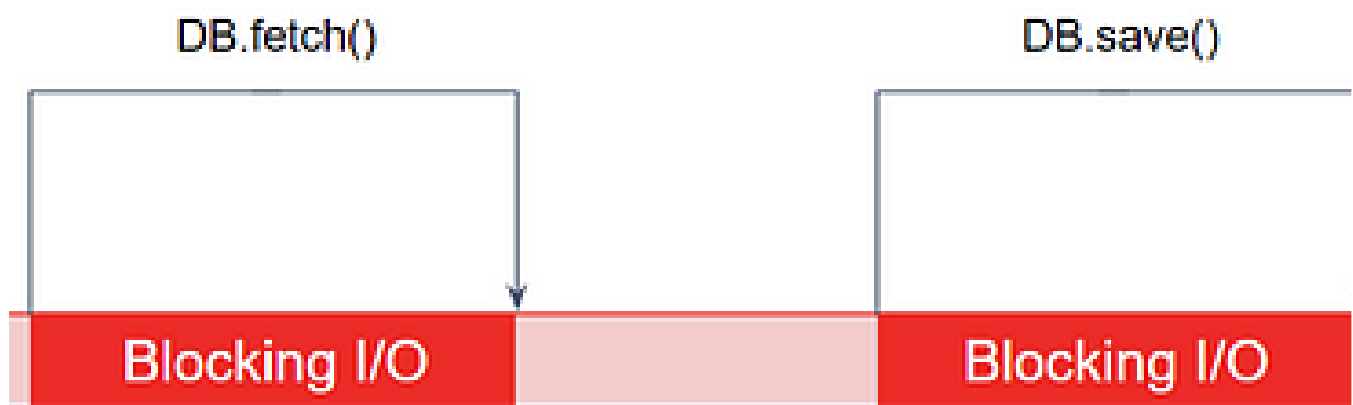
8 min read · May 13, 2024




383



2



 Visarut Sae-Pueng in Ascend Developers

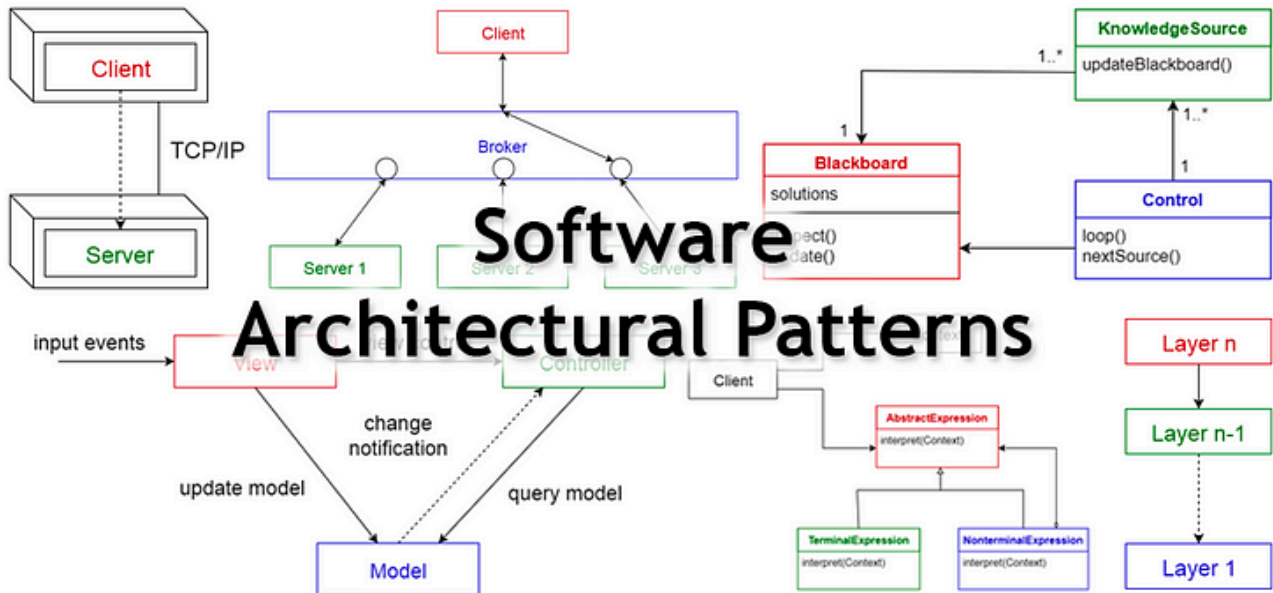
## Why are my JAVA virtual threads slower than the platform threads?

The problem with platform threads is the wait for I/O operations to complete. That's essentially idle as the thread is unable to perform...

6 min read · May 4, 2024

👏 198    💬 4

🔖    ⋮



hubian

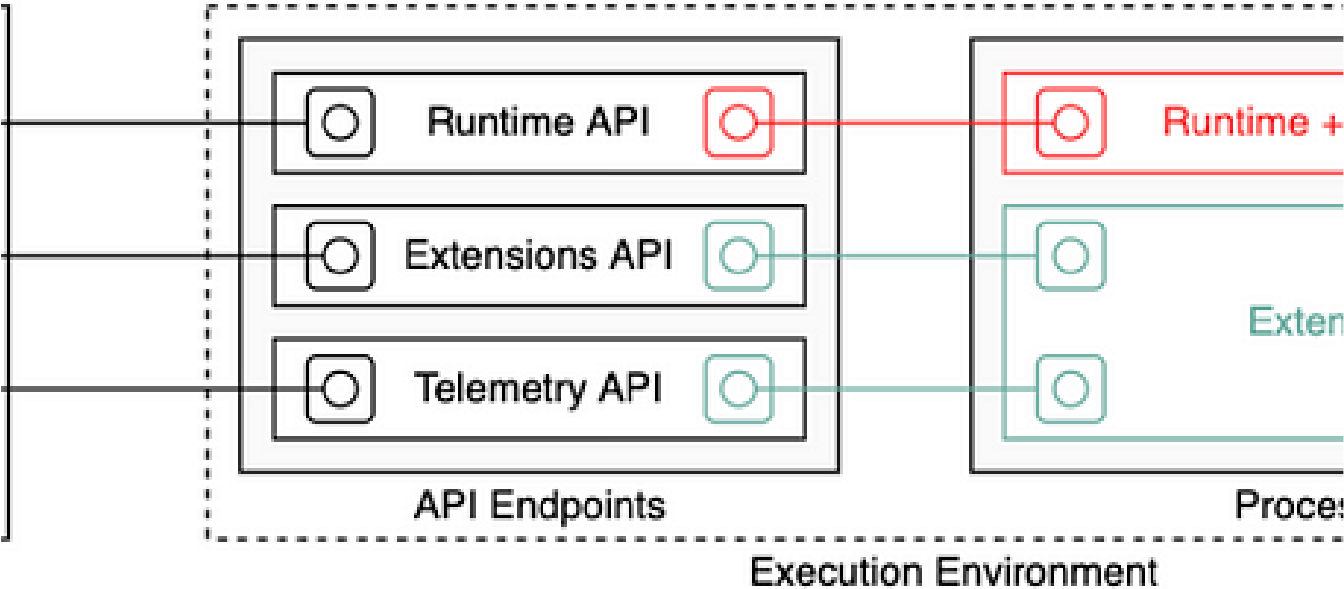
## 12 common software architecture styles, essential for architects


What is software architecture?

14 min read · Jan 8, 2024

👏 53    💬 1

🔖    ⋮



 ifezouani ilyass in AWS Tip

# Advanced AWS Lambda—An Obscure Feature You Must Absolutely Use

Introduction

7 min read · Apr 18, 2024

 193     2        

See more recommendations