

How I built a streaming backend

ZePlay: adaptive HLS, background transcoding, and what load testing taught me.



- FastAPI
- FFmpeg
- HLS.js
- Redis
- PostgreSQL
- React

Syed Asjad Abbas

AI/ML Engineer · Founder & Lead Engineer, Zeploy Tech

Swipe →

WHY NOT JUST SERVE THE FILE?

One big MP4 **doesn't scale**

Slow connection

constant buffering

Seeking ahead

downloads what you skip

Encode in a request

blocks the server

ZePlay was my attempt to build the systems underneath a streaming service, not copy its interface.

DECIDED ON DAY ONE

Priorities **before** features

1 Availability

2 Scalability

3 Speed

4 Security

5 Streaming performance

6 **Features** last, on purpose

A feature wasn't done until its effect on the first five was checked.

Two paths, kept apart

UPLOAD PATH

Admin upload

source video



FastAPI

saves job, replies at once



Redis queue

job, lock, retries



Worker + FFmpeg

3 renditions → HLS

PLAYBACK PATH

Viewer

React + HLS.js



Entitlement check

plan verified on server



HLS storage

playlists + segments



Adaptive playback

quality follows bandwidth

Slow work happens where viewers can't feel it.

BACKGROUND PROCESSING

Transcode **off** the request path

The API saves the job and answers at once. A worker pulls it from a Redis queue, takes a lock, and retries on failure.

```
# one lock per video, 10-minute expiry  
SET zeplay:lock:job:<id> worker NX EX 600  
# → no video is encoded twice
```

ADAPTIVE STREAMING

Three renditions, **six-second** segments

HLS.js switches quality as bandwidth changes. Seeking fetches only the segments near the new position.



Network dips, quality drops, playback keeps going.

PREMIUM CONTENT

Security lives on **the server**

Every playlist and segment request checks the viewer's plan. Hiding a button in the UI is not a barrier.

Not signed in	denied
Free plan, premium title	denied
Entitled viewer	stream

LOAD TESTING

Zero failures can still be **too slow**

Every request succeeding says nothing about how long people waited. Track **median and P95**, not just errors.

Failure rate	can read 0%
Median latency	the typical wait
P95 latency	the real experience

And test with the cache switched off. A cache hides slow queries until the day it misses.

Syed Asjad Abbas

AI/ML Engineer

Swipe →

SCALING

Scaling moves **the bottleneck**

More API workers need a database connection budget. Otherwise PostgreSQL becomes the new limit.



THE HONEST ONE

Not every optimization survives

A faster pagination approach looked great in isolation. I rolled it back to keep playback stable.

Cursor pagination	fast on its own
Playback stability	non-negotiable
Decision	roll back, retest

An optimization only counts if it works with the whole system. The next version brings it back, behind tests.

Syed Asjad Abbas

AI/ML Engineer

Swipe →

ZePlay in 8 lines

- 01 Set priorities before features
- 02 Keep upload and playback apart
- 03 Transcode in a worker, not a request
- 04 3 renditions, ~6 s segments
- 05 Enforce access on the server
- 06 Measure P95, test with cache off
- 07 Budget DB connections first
- 08 Keep what holds up under load

The code is **open**

Architecture, worker, HLS pipeline and tests are all in the repository.

github.com/syedasjadabbas/Zeplay

Follow for more on AI/ML and backend engineering.

Syed Asjad Abbas

AI/ML Engineer · Founder & Lead Engineer, Zeploy Tech