LDX3 NYC 2026

One shot at scale.

Surviving the Super Bowl signup surge: engineering decisions made under a deadline you cannot move.

VU

Vadim Uchitel
Principal Engineer, Fetch
30 years in software · 20 years building systems for NASA

THE SETUP · SUPER BOWL LIX

The Big Reward, to the Big Game.

A live video stream inside the app during the final two minutes of the game. A live drawing on top of it.

01 **$1.2M live sweepstakes.** 120 winners, $10,000 each, drawn during the final 2 minutes of the game.

02 **Open the app, you're entered.** No form, no QR scan, no opt-in screen. The lowest possible bar.

03 **Winners announced in-app, live.** Names appearing on millions of screens in sync with the broadcast.

120 football seconds. One shot. Legally binding.

THE GAP WE HAD TO CLOSE

1

Normal Tuesday signup load (signups/sec)

150,000

Peak signup target (signups/sec)

Five orders of magnitude. Instantaneously.
The moment our Super Bowl commercial aired.

BEFORE WE WROTE A LINE OF CODE

The constraints, on day one.

103

days from "what if we did something crazy?" to kickoff.

6

engineers on the build team. BE, iOS, Android, 2× FE, plus DevOps.

1

immovable date. February 9, 2025. Super Bowl LIX kickoff.

SLS LAUNCH · NASA

Why is it hard to launch satellites?

~~Gravity is solved.~~

~~Physics is solved.~~

You get one shot.

The Super Bowl was our satellite launch. Once the commercial started, we could not roll back.

SETUP · WHY THIS IS DIFFERENT

This is a scale talk, but it's not a normal one.

NORMAL SCALE STORY

Drop a few requests. Retry. Apologize on Social Media.

Most launches let you trade availability for graceful degradation. The escape hatch is built in.

Has an escape hatch

LEGALLY BINDING SWEEPSTAKES

$1.2M live sweepstakes. Every entrant counted. Every winner provably valid.

Some operations had to be both fast and provably correct. That changes how you decompose a critical path.

No escape hatch

Backends can't absorb millions of clients at once. CDNs can.

APP SIDE

Embedded webview

Native UI swapped for a static site behind a CDN. The livestream never touches our backend.

SYNC MECHANISM

metadata.json

Single static file polled every 4s. Carried winners, CDN weights, rollout percentage. The CDN is the API.

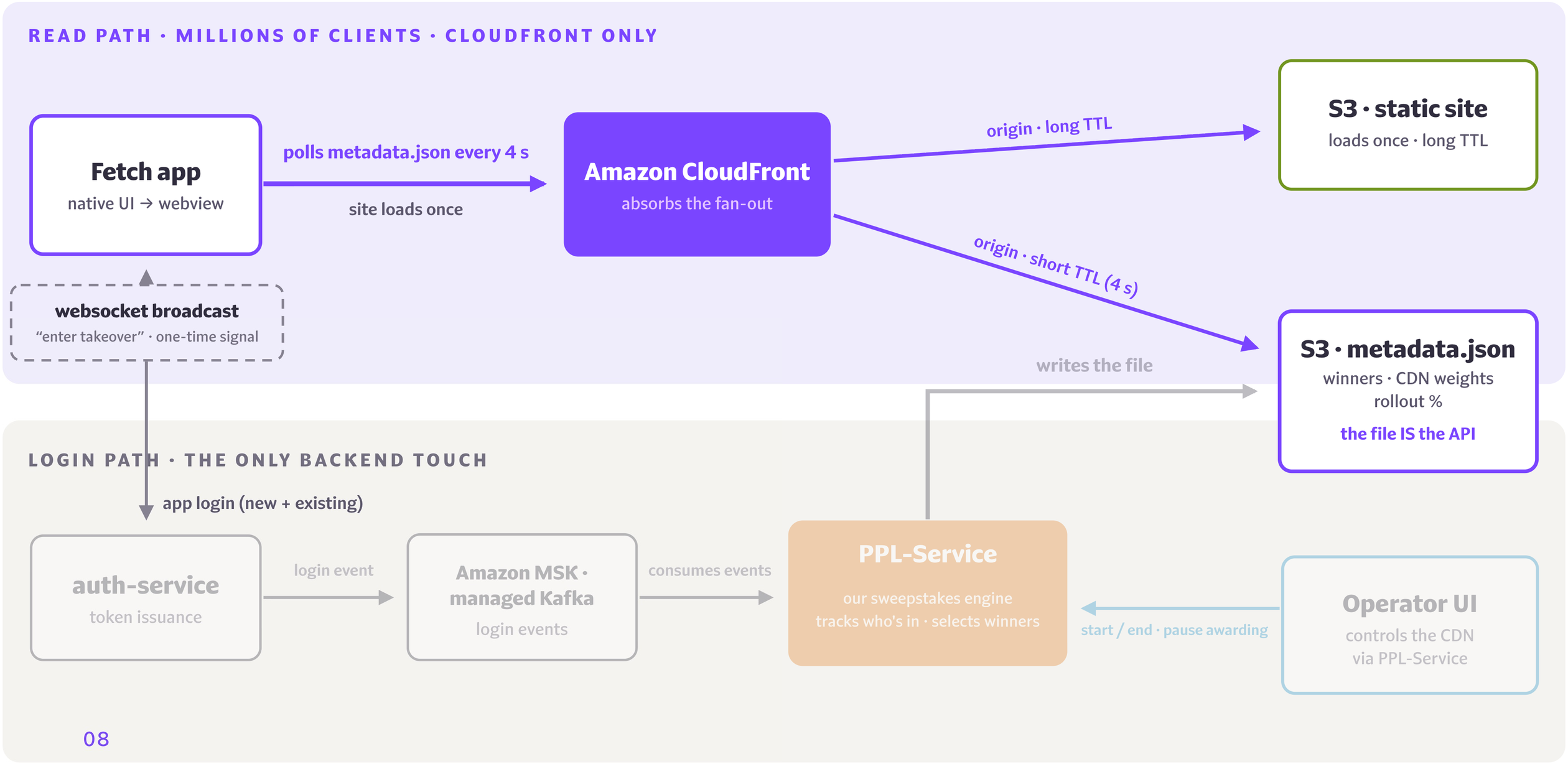
USER SIDE

Open the app, you're in.

No "how do I get into this?" Tap the icon at kickoff and the livestream is right there. The CDN absorbs millions in parallel.

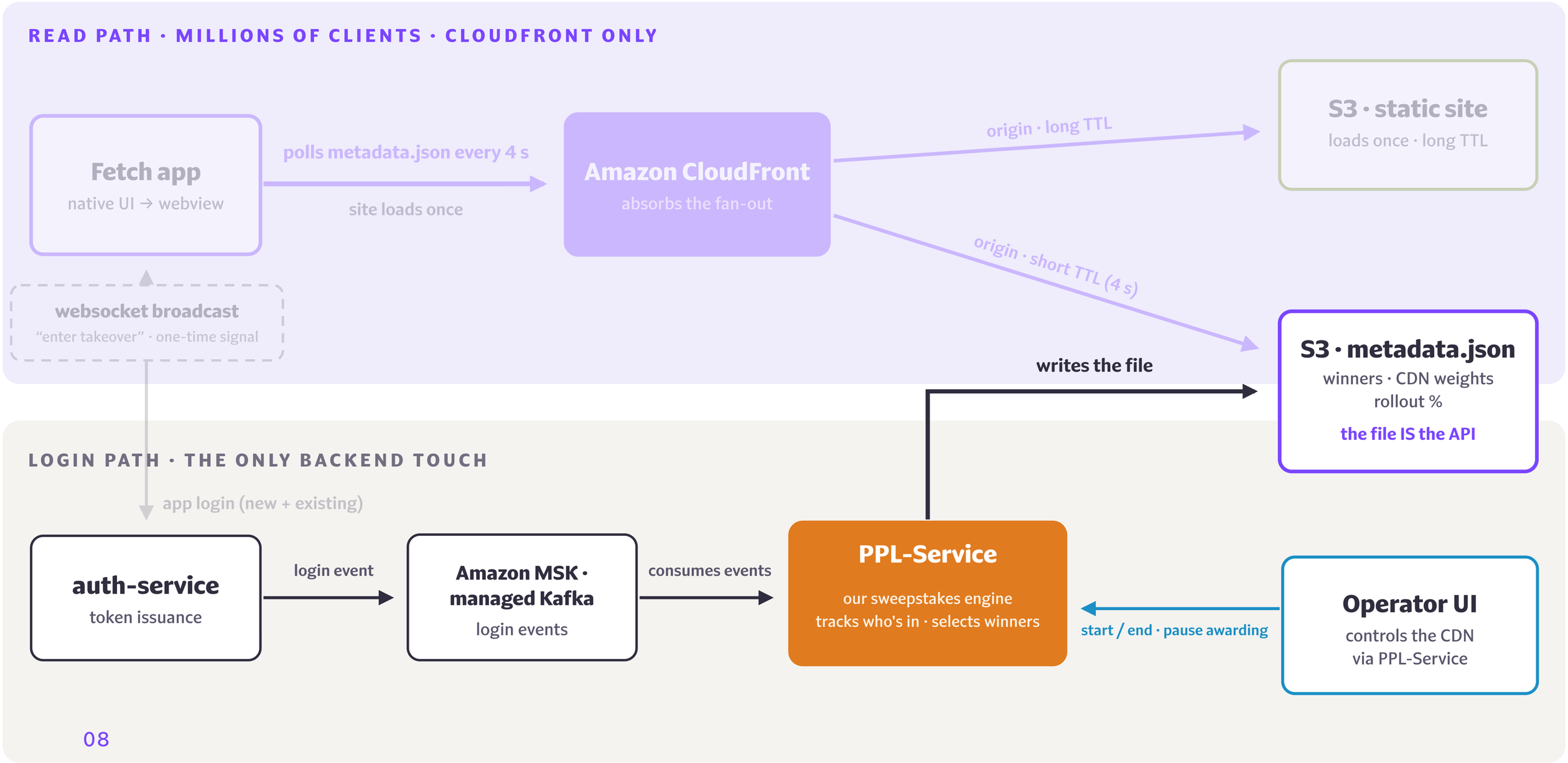
TAKEOVER MODE · THE ARCHITECTURE

CloudFront in the middle. Backend nowhere on the hot path.



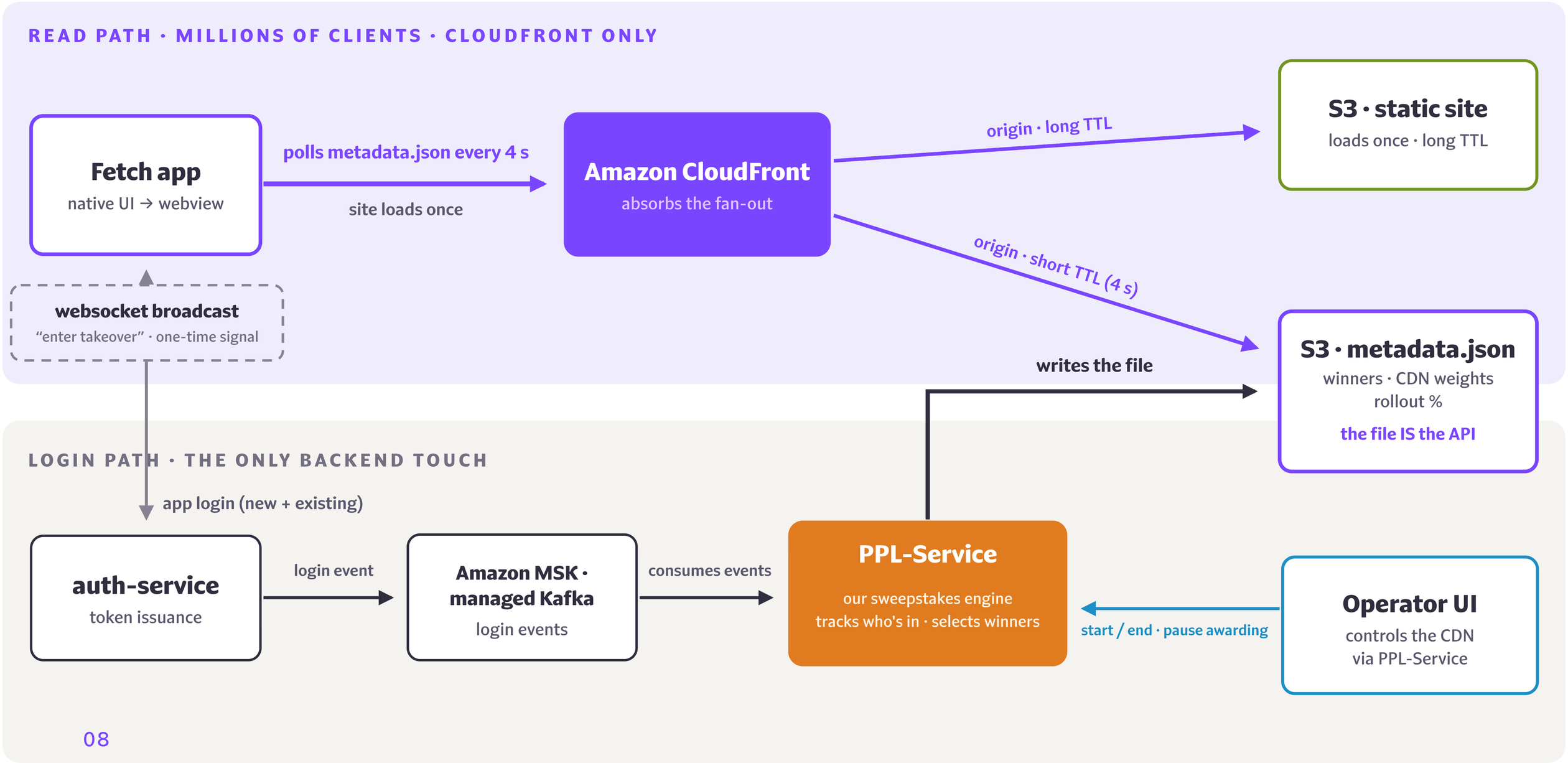
TAKEOVER MODE · THE ARCHITECTURE

CloudFront in the middle. Backend nowhere on the hot path.



TAKEOVER MODE · THE ARCHITECTURE

CloudFront in the middle. Backend nowhere on the hot path.



TWO PATHS IN · ONE PATH OUT

Existing users walked in. New users had to be created first.

PATH A · EXISTING USERS

Tap the app, you're in.

Open app → Auth (cached) → Takeover (CDN)

Already provisioned. Token is fresh. Skip the queue, hit the CDN.

PATH B · NEW USERS

Create them, then let them in.

Open app → Sign up → Identity + auth + verify → Takeover (CDN)

Scaling *this* path is the rest of this talk.

Once inside takeover mode, both paths look identical: a webview polling a static `metadata.json`. **Zero backend communication. The CDN is the experience.**

THE UNMODIFIED HOT PATH

What "create a user" actually does.

APP LAUNCH

semaphore-service
Remote config + translated strings (via multi-region gateway)

SSO provider
Google · Apple

SMS provider
Verification codes via AWS SMS

USER RECORD CREATION

identity-service
Postgres write, the user record itself

auth-service
Token issuance + storage

device-service
Device record

VERIFICATION STACK

device dedup
One account per device

app version
Block deprecated builds

email dedup
No duplicate addresses

IP rate limit
Throttle rapid signups from one IP

THE FRAMEWORK

For every operation on the critical path, one question: which bucket?

BUCKET 01

Must be synchronous

Has to happen before the user moves on. Cannot be queued, cannot be skipped.

"Does the user need this answered now?"

BUCKET 02

Can be deferred

Real work, but the user doesn't need it complete to continue. Move it off the hot path via Kafka or SQS.

"Can a queue eat this?"

BUCKET 03

Can be dropped

Remove it from the flow entirely for the duration of the event. Behind feature flags, ready in seconds.

"Can we live without this for 2 minutes?"

We could test the load.
We couldn't test the world.

WHAT IT COULD DO

150,000+ signups/sec against the Fetch-owned backend stack.

Pre-warmed ALBs, horizontally-scaled ECS tasks, real Kafka, real DynamoDB, real Postgres. We pushed until something fell over, fixed it, repeated.

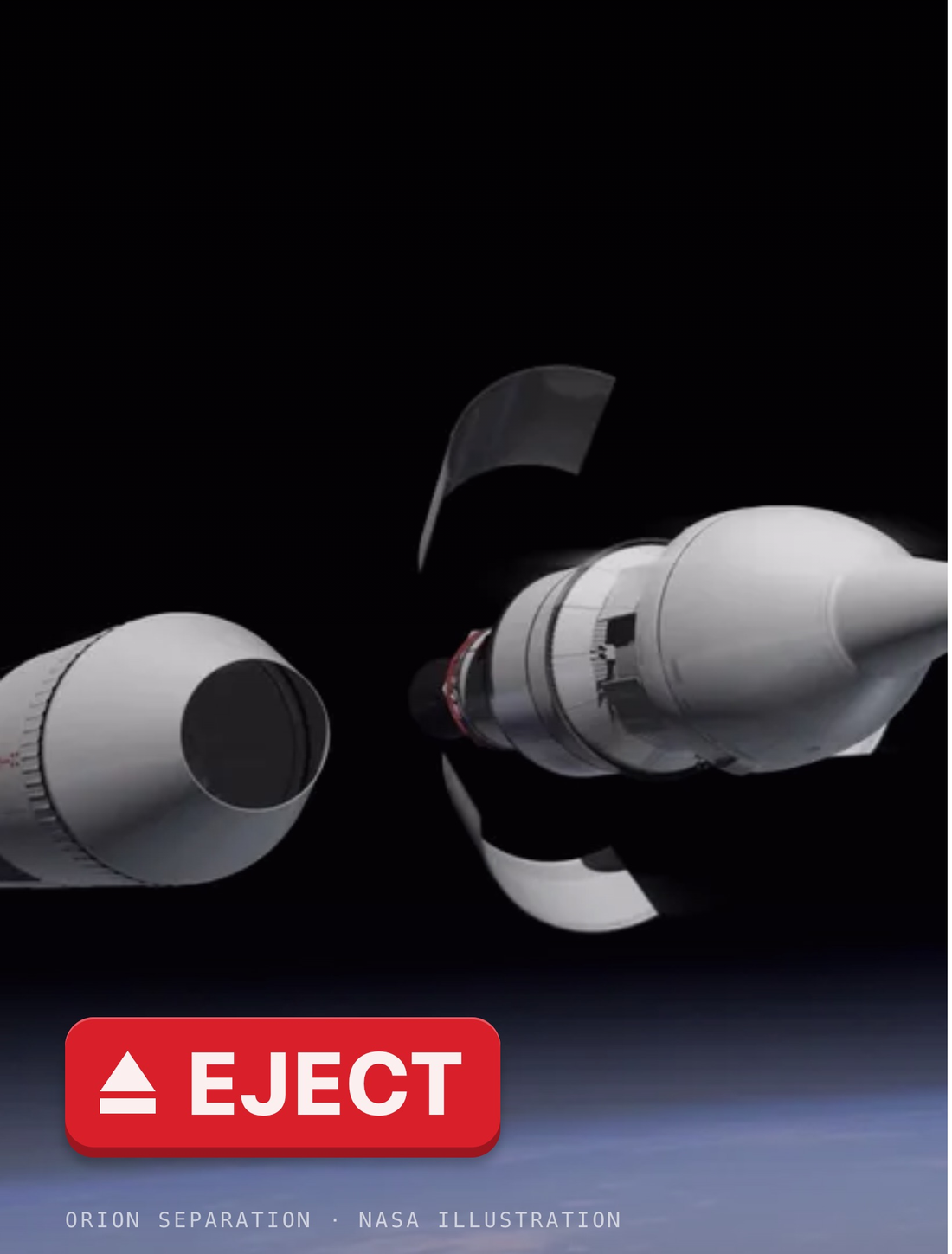
Methodically increased load until failure

WHAT IT COULDN'T DO

Call Google. Call Apple. Send real SMS.

Third-party APIs got simulated with a `Sleep`. Educated guesses at latency. The simulator never failed, never throttled, never said no.

Three places where reality could bite. Remember this slide.



ORION SEPARATION · NASA ILLUSTRATION

BUCKET 03 · DROP ENTIRELY

What we dropped for 120 football seconds.

01 **Demographics screens.** Multi-step signup collapsed to one screen. Removed phone-number requirement for SSO users.

02 **Phone signup, demoted.** Moved to the bottom of the screen. Behind a remote config flag, ready to disable instantly.

03 **Verification checks, flagged off.** Device dedup, email dedup, IP rate limiting, all behind feature flags, ready to relax under load.

04 **Non-essential SQS consumers, paused.** Geolocation enrichment and similar background work, paused during the game to give Postgres breathing room.

The trade: eligibility stays fixed. Cleanup is recoverable.
A meltdown is not.

BUCKET 02 · MOVE OFF THE HOT PATH

What we deferred: synchronous waits became queue writes.

01 **device-service → Kafka.** Stopped writing the device record synchronously. Event in, processed later.

02 **identity-service & auth-service → SQS.** The durable writes (DynamoDB, Postgres) were buffered and drained after the game. The Redis entry write stayed synchronous.

03 **Phone verification → dedicated Redis cluster.** Off the shared one. Isolated blast radius.

04 **Token expiration extended.** Pre-game logins got tokens valid until 3 AM after kickoff, not the usual 1 hour. Existing users walked in with a live token. No auth-service hit.

What we couldn't drop and couldn't defer.

01 **SSO verification.** If a user signs in with Google or Apple, we have to actually call Google or Apple. Cannot fake that.

02 **SMS verification.** When a user signs up with a phone number, we cannot defer the code. They are staring at the screen waiting.

03 **The eligibility commit.** The database row could catch up later. A drawable record and durable event had to exist before the user could win. Legal correctness.

04 **The auth token.** The user can't do anything in the app without it. Cannot queue a token.

Most of our scaling effort went here. Fast *and* auditable, at full target load.

SURPRISE · THE BOTTLENECK WE DID NOT EXPECT

Your gut about which thing breaks first is probably wrong.

EXPECTED CEILING

DynamoDB

Held steady at 150K RPS.

For small-object storage, throughput stayed flat. We never hit an upper limit during testing. The database we were nervous about quietly did its job.

ACTUAL CEILING

Redis

CPU-bound above 95%. Calls took seconds.

Single-threaded per shard. Even our largest instance pinned. We redesigned around dedicated clusters, sharding, smaller hot keys, before game day, not during.

JANUARY 15 · THE REHEARSAL WE COULD RUN

We ran the event in production, four weeks early.

70K

peak concurrent viewers. Less than 1% of the 20M+ we were bracing for.

Push → 180 winners → 100K Fetch points each. Real prize. Real load.

WHAT IT CAUGHT

An iOS video-player lockup.

Swapped the player's rendering engine. That fix alone paid for the rehearsal ten times over.

WHAT IT COULDN'T TEST

The signup surge. The rehearsal only exercised the takeover flow for existing users.

A new-user surge only happens once, when a Super Bowl commercial airs. There is no dress rehearsal for that.

GAME DAY · THE ACTUAL NUMBERS

The story the dashboards told.

2.3M

concurrent livestream viewers at peak.

~1.1M

brand-new signups during the event window.

3 min

to return 100% of viewers to the regular app, with zero rollout failures.

$1.2M

prize pool, drawn live, every winner provably valid.

Signup load spiked in the first 30 seconds, then came in below our 150,000 signup/sec design target.

SURPRISE LIVE · THE SMS STORY

A known limit. A single Zoom. A real cost.

COMMERCIAL AIRS

App goes into takeover.
Signup surge starts.

WITHIN 30 SECONDS · SMS LIMIT HIT

AWS 1,000 RPS hard cap.
Phone signups start failing.

? MINUTES · ZOOM STOOD UP

Pulling in the right approvers cost more time than the fix.

We don't know how long this took. Nobody was timing it. That's part of the problem.

A FEW MINUTES LATER · PHONE OPTION FLAGGED OFF

Remote config switch flips.
Errors stop instantly.

The technical fix would have taken minutes. **A single Zoom call became the bottleneck of the entire system.**

Phone signups failed, but entry never closed. SSO worked the whole time. How many gave up instead of retrying? We don't know.

Five offices. One Zoom line.

MADISON, WI

- **Mobile dev · phone-flag owner**
- Backend
- FE

CHICAGO, IL

- FE
- Mobile
- DevOps

BIRMINGHAM, AL

- **Me · backend, on ops**
- FE
- Data

BOSTON, MA

- Backend
- Mobile
- Leadership

NYC

- Backend
- FE
- Data

Roles were scattered, partially by accident. Every office had a mix.

- - - - A SINGLE ZOOM LINE - - - -

Everyone on it. Multiple conversations crossing each other. SMS limit trips → I spot it in Birmingham → interrupt the line → wait to be heard → find the right mobile dev (Madison) → approval (Boston) → flip the flag. Precious seconds, mostly spent talking over each other.

THE FIX · DESIGN THE DECISION PATH LIKE YOU DESIGN THE SYSTEM

Named owner
whose hand is on the flag

→

Decision threshold
the number that means “flip it now”

→

Approver
decided before kickoff, not during

One dedicated action channel per feature team. The shared line is for context. It is not for action.

EXIT · THE SAME LEVER BOTH WAYS

2.3M users back into the app, without melting anything.

Operator UI slider
A single integer. Updated by hand, in real time, by one operator.

→

metadata.json
Already on the CDN. Adds `rolloutPercentage` field. 4-second cache TTL.

→

Client webview
Hash userId. If under threshold, send "exit takeover" message to native shell.

The same file that announced the winners drained the building. Dialed gradually over 3 minutes. No backend ever saw a thundering herd.

RE-ENTRY

The mission succeeded. Then the capsule hit the atmosphere.

ORION RE-ENTRY · NASA ILLUSTRATION

POST-GAME · THE POSTGRES INCIDENT

The game ended. The crisis didn't.

AFTER THE GAME	**Users opened their profiles to update info. Many added phone numbers.**
SERVER FLEET STILL SCALED	**Identity-service still horizontally scaled. Every server carrying its own connection pool.**
CEILING CROSSED	**Connection count crossed RDS Postgres' ~5,000 ceiling. Instance unresponsive. Database reboot.**

We tested query throughput. **We never tested the server fleet.**

IN HINDSIGHT

Three things, if I'm doing this again.

01 **Rehearse the failure, not just the fact.** We knew the 1,000 RPS SMS cap. Our simulator slept instead of throttling, so no stress run ever exercised the error path. The first kill-switch flip was live.

02 **Load-test the server fleet, not just the query.** Postgres query throughput was fine. Server count × connection-pool size crossed the ceiling. Invisible at smaller fleet sizes.

03 **Rehearse communication, not just systems.** We load-tested the architecture. We didn't load-test the people on the single Zoom line.

IF YOU'RE STARING DOWN AN IMMOVABLE DEADLINE

Four things to take with you.

01

Decompose the critical path.

Synchronous, deferred, droppable. Feature flags and queues are scaling levers, not optimizations.

02

Prepare to be wrong.

Our gut said DynamoDB. Reality said Redis. Test the assumption, don't defend it.

03

Team operations are architecture.

Your org chart is on the critical path. A decision that threads five offices is a single point of failure, same as any service.

04

Ready means rehearsed.

Everyone has a risk register until production punches them in the mouth. Awareness is the cheapest, weakest form of preparation.

Thank you.

Vadim Uchitel
Principal Engineer · Fetch

linkedin.com/in/vadimuchitel

ORION SPLASHDOWN · NASA

LDX3 NYC
2026