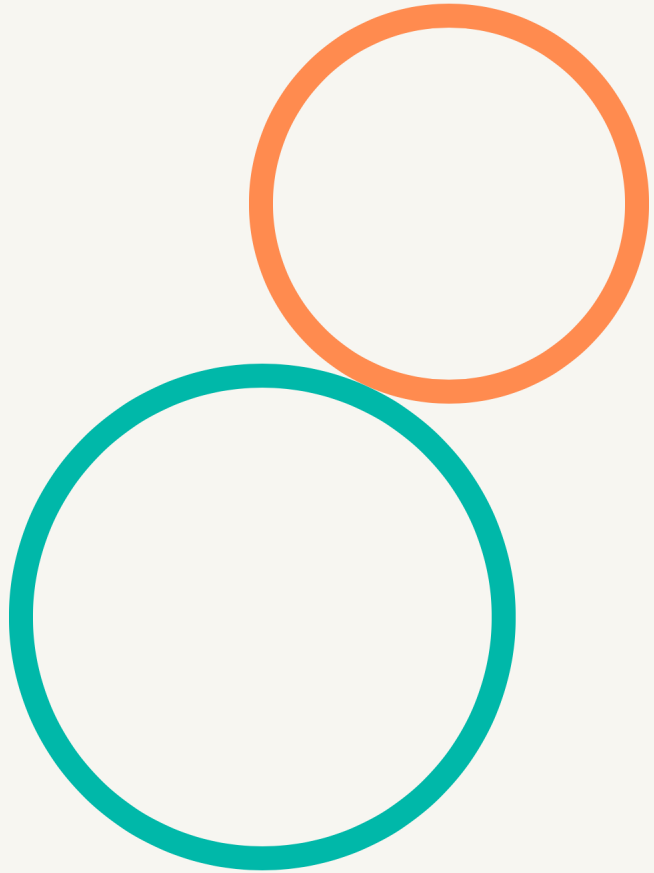
qualabs

Observability Day · KubeCon + CloudNativeCon North America 2026

Observing the Stream

Open source monitoring for video infrastructure

Erika Álvarez · DevOps Engineer, Qualabs





Hi, I'm
Erika Álvarez

DevOps Engineer at Qualabs
Montevideo, Uruguay

We build and run video solutions for broadcasters and streaming platforms

WHAT WE'LL COVER

1 The problem we kept hitting

2 The open source stack we built

3 How it runs: Git to cluster

4 What we learned, honestly

THE PROBLEM

Every dashboard was green. The picture was black.

Nothing crashed. Nothing threw an error. The stream just degraded.

THE PROBLEM

Before: flying blind

Errors caught late

No visibility into live and on-demand video workflows.

Incidents escalated unnoticed

By the time someone complained, the moment was already gone.

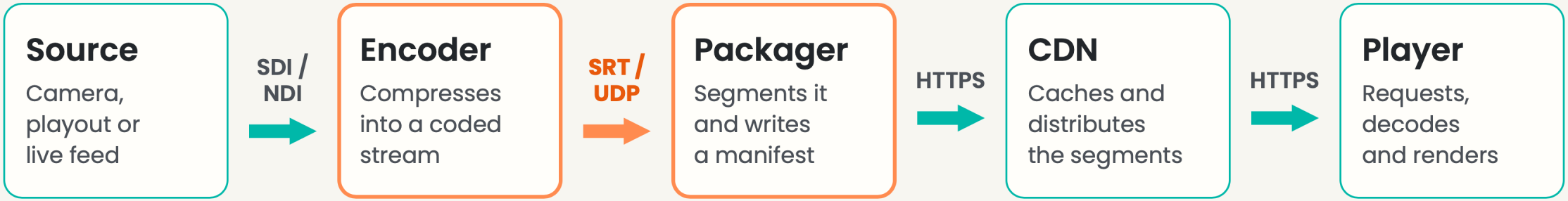
Commercial tools too expensive

Commercial monitoring tools were out of reach for us.

The first hour of every incident went to working out what was wrong.

VIDEO 101, FOR OBSERVABILITY PEOPLE

Streaming in one slide



Orange is the leg we monitor in this talk:
The live signal between Encoder and Packager, before anything is packaged.

WHY GENERIC MONITORING MISSED IT

Healthy infrastructure, broken video

False health

Infrastructure can look perfectly healthy while the picture is black.

Blind spots

CPU, memory and HTTP 200s say nothing about whether the video is watchable.

Silent failure

Nothing crashes, nothing throws. The stream just degrades.

Delayed alarm

By the time someone complains, the moment is already gone.

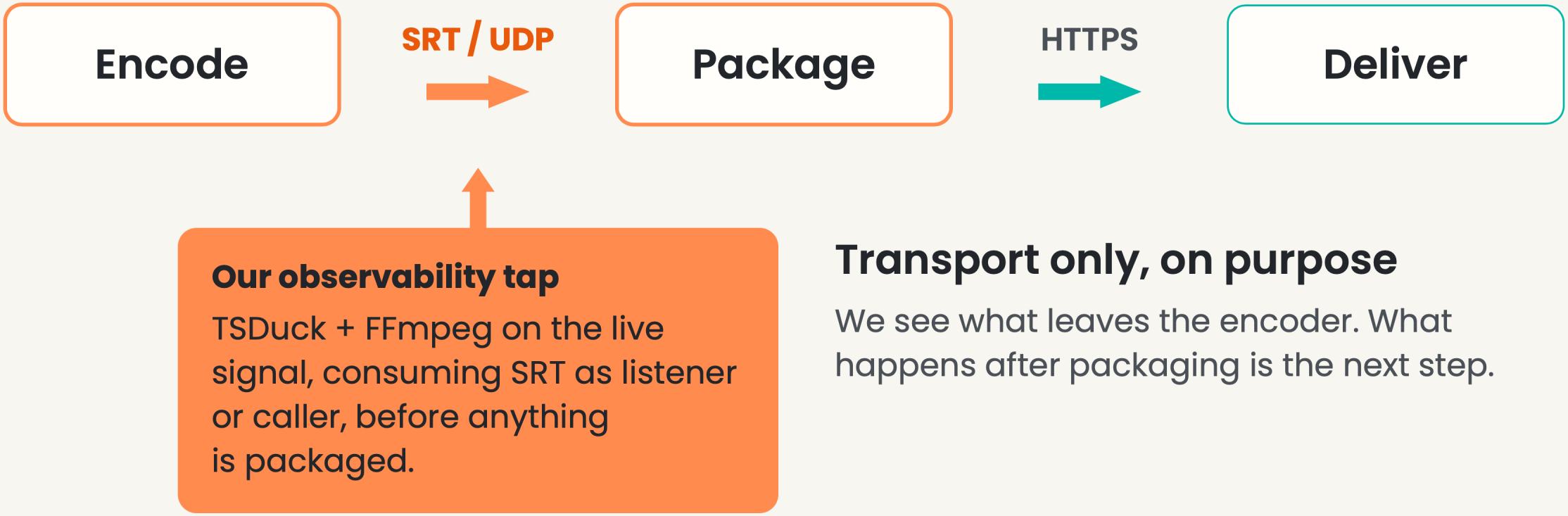
THE DECISION

Why we built our own

Cost	Commercial monitoring tools were too expensive.
Depth	We needed signals from the stream itself, at the transport layer, not just infrastructure metrics.
Proof	Broadcasters hold us to quality commitments we couldn’t measure.
Ownership	A maintainable open source solution we can tune and deploy like any other service.

THE SOLUTION

Where the monitoring taps in



Transport only, on purpose

We see what leaves the encoder. What happens after packaging is the next step.

THE SOLUTION

The stack: open source, one job each

Extract the signals

Collect, store, visualize and alert



All of it runs on Kubernetes, packaged with Helm and deployed by Argo CD.

THE SOLUTION

Two layers of signal

Transport layer

Reads the stream packet by packet.

- PIDs
- Continuity counters
- PCR timing
- ETR-290 compliance

Content layer

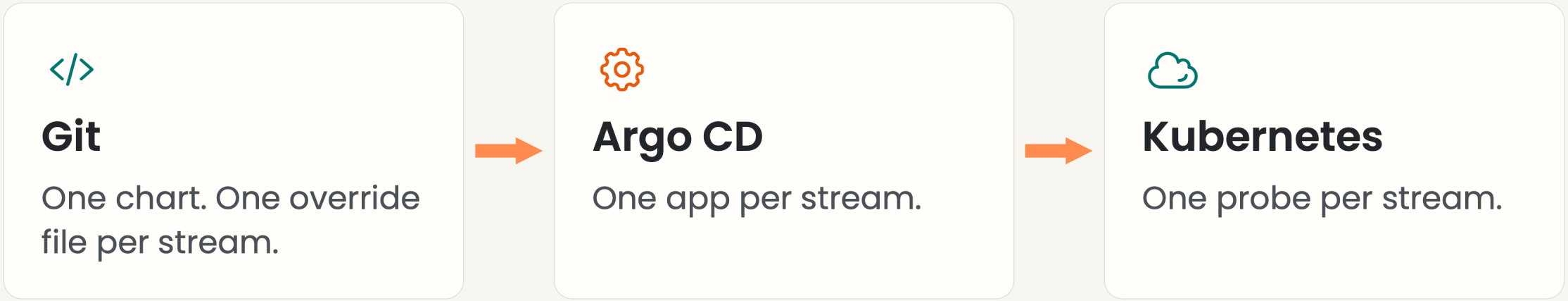
Decodes samples of the same signal to see what is actually in it.

- Black frames
- Silence
- Frame-rate drift

A custom Python exporter turns both into Prometheus metrics.

RUNNING IT IN PRODUCTION

How it runs: Git to cluster



RUNNING IT IN PRODUCTION

Onboarding a new stream

1

Add one file to Git

Stream name, input and any threshold overrides.

2

Argo CD creates the probe

ApplicationSet: one probe per file.

3

Prometheus discovers it

Found automatically, no Prometheus change.

4

The dashboard shows it

One dashboard with a stream variable.

5

Alerts apply by stream label

Generic rules, overrides only when needed.

WHAT IT GIVES US

What it catches

What we detect	What it could mean for the viewer
Continuity errors	Glitches, pixelation or a brief freeze
PCR drift	Audio and video drifting apart, or stalling
A PID goes silent	Picture or sound gone entirely
Black frames or silence	A lower quality experience

Real-time visibility, per stream

WHAT WE LEARNED

Thresholds that hold

Baseline first

Watch a healthy stream for days before deciding what abnormal looks like.

Rates, not single events

One continuity error is noise. A sustained rate is a problem.

Tier by impact

ETR-290 priorities map well: P1 pages someone, P3 stays on the dashboard.

Tune from real incidents

Every false positive and every miss is feedback on the threshold.

OUTCOMES

What changed

Earlier error detection

Time to detect a broken stream, went down.

Real-time visibility

Streams monitored, per stream, as they happen.

Costly tools replaced

Commercial tooling swapped for open source we can maintain.

TAKEAWAYS

Three things to take home

1. You have to deeply understand your system to monitor it properly.
2. A custom stack is worth designing for specific or non-conventional workflows.
3. The tool is the easy part. Deciding what counts as broken is the work.

Thank you!

Questions?



Erika Álvarez

DevOps Engineer at Qualabs

qualabs