Creating FAST Channels with Open Source Tools

How VOD2Live technology and the Eyevinn open source FAST stack let you launch, program, and monetize linear channels at a fraction of the traditional cost and energy footprint.

EXECUTIVE SUMMARY

FAST channels (Free Ad-supported Streaming TV) offer viewers a lean-back, linear TV experience without a subscription, and offer content owners a way to monetize libraries that would otherwise sit idle. The economics, however, only work if channels are cheap to launch and cheap to run. A channel that needs a 24/7 live transcoding chain can never be a long-tail channel.

This paper describes how to create FAST channels with **VOD2Live** technology and the open source Eyevinn FAST stack: linear HLS channels produced from already transcoded VOD assets, programmed by a static playlist, a schedule, or fully dynamic just-in-time logic, and prepared for ad monetization with standard signalling that works with the server-side ad inserter of your choice. Every component is open source, and every component is also available as a managed service in Eyevinn Open Source Cloud, so a channel can be launched in minutes and moved in-house later without re-platforming.

THE FAST OPPORTUNITY, AND ITS COST PROBLEM

A successful FAST strategy involves assembling linear content streams, ensuring high-quality delivery, and dynamically inserting ads, all while keeping the flexibility to change programming quickly. The catch is volume: FAST portfolios succeed by offering many targeted channels, and each traditionally built channel carries the cost of a continuous live encoding and transcoding chain, running around the clock whether anyone is watching or not.

That model burns money and energy on re-encoding content that has already been encoded once, as VOD. For a library owner considering ten or fifty niche channels, per-channel transcoding is usually the line item that kills the business case.

VOD2LIVE: THE CORE IDEA

VOD2Live turns the problem around. Instead of playing content into an encoder and transcoding a live signal 24/7, the channel is assembled at the manifest level from **already transcoded HLS VOD assets**. A channel engine stitches the VOD manifests into a continuous linear HLS stream that any HLS-capable player and CDN can handle like an ordinary live channel.

> ***The channel inherits the video and audio quality of your VOD encodes, at a fraction of the energy and cost of a live transcoded channel.***

Because no media is re-encoded, quality is whatever your VOD pipeline produces, including your best encodes. The approach supports the common HLS flavors (including CMAF and demuxed audio and video), WebVTT subtitles, and DRM, and channels can mix VOD2Live content with genuine live HLS streams for events. It even enables personalized channels, where each viewer gets a unique 24/7 stream.

THE EYEVINN FAST STACK

Eyevinn has built and open sourced the components needed to go from a content library to a monetizable channel portfolio. All of them are also available as managed services in Eyevinn Open Source Cloud (OSC) at osaas.io:

- **Channel Engine** (Apache 2.0 library): the core VOD2Live engine. It deliberately contains no business logic; you connect it to adapters that tell it what to play next, which keeps programming decisions fully in your hands.
- **FAST Channel Engine** (prebuilt service and container): the Channel Engine bundled with ready-made programming plugins (Loop, Playlist, Schedule Service, and WebHook), so you can run channels without writing code.

- **Schedule Service**: channel and schedule management with an API, including an auto-scheduler that populates a channel from a playlist and support for calendar-style programming.
- **HLS VOD Stitcher**: a proxy that inserts other VODs (such as ad placeholders) into a source VOD at instructed positions, controlled by a simple JSON payload.
- **Supporting services**: a multiview UI for monitoring channel portfolios, a serverless function service for hosting programming logic, and an open source test ad server for validating the ad chain end to end.

THREE WAYS TO PROGRAM A CHANNEL

The stack supports three programming models of increasing sophistication. You can start with the simplest and upgrade later; the streaming URL and the viewer experience stay the same.

MODEL	HOW IT WORKS	BEST FOR
Playlist	A text file with one HLS VOD URL per line; the engine plays the list on repeat. A loop mode plays a single VOD continuously.	Pop-up channels, single-title channels, quick pilots
Scheduled	The Schedule Service holds channels and schedule events behind an API; an auto-scheduler can populate channels from playlists, and editors can program by calendar.	Curated channels with editorial planning and EPG needs
Dynamic	The engine calls a webhook when it is about to switch to the next VOD; your code decides just-in-time what plays next, per channel. The logic can run as a small serverless function in OSC.	Data-driven, personalized, or rights-aware programming

The dynamic model deserves emphasis. The webhook receives the channel id and returns the URL of the next VOD, optionally with a preroll. Whether the decision comes from an editorial system, a recommendation engine, or a rights window database is invisible to the engine. Programming becomes a small piece of code you own, not a feature you wait for a vendor to ship.

AI AND METADATA: CHANNELS THAT CURATE THEMSELVES

If transcoding was the first cost barrier for FAST portfolios, curation is the second. A channel is only as good as its programming, and hand-scheduling fifty niche channels does not scale. The combination of rich metadata and AI turns curation from an editorial bottleneck into a service.

The foundation is metadata. Catalog metadata (genre, cast, era, language, rights windows) already supports rule-based curation. AI extends it: speech-to-text and subtitle generation make dialogue searchable, scene and topic detection describe what actually happens in the content, and embeddings make the library searchable by theme and mood rather than only by tags. The result is a library where a channel concept can be expressed as a query.

The stack has two natural places to plug this in. In the **scheduled model**, an AI planner writes schedule events into the Schedule Service through its API, filling channels days ahead while editors review and adjust in the same interface. In the **dynamic model**, the AI curator sits behind the webhook and picks the next asset just-in-time, using the channel concept, what has recently played, time of day, and any signal you choose to feed it. The engine does not change at all; curation is simply the code you already own behind the webhook.

This is what makes **pop-up channels** practical. Define a channel as a concept: a director retrospective, a festival, a sports event weekend. The AI assembles a candidate list from the metadata, hard filters enforce rights and windows, an editor approves, and the channel is live the same day. When the event passes, the channel is retired. With per-channel costs this low, a channel can exist because a weekend justifies it.

The guardrails matter as much as the intelligence: rights and licensing windows are hard filters the AI cannot override, brand and editorial rules constrain what may be adjacent to what, and human approval remains in the loop where the channel warrants it. AI proposes; your rules and your editors decide.

BUILT-IN MONETIZATION READINESS

A FAST channel is only a business if it can carry ads. The stack prepares channels for monetization without binding you to any ad vendor:

- **Ad placement opportunities.** The HLS VOD Stitcher inserts house ads into the source VOD at chosen positions and durations, defined in a JSON payload. Each insertion is signalled in the stream as a replaceable break.

- **Server-side ad insertion (SSAI).** A downstream server-side ad inserter (for example Yospace, AWS MediaTailor, or Nowtilus) replaces the signalled house ads with targeted ads decided by the ad server, per user or session.
- **Server-guided ad insertion (SGAI).** The Channel Engine also supports HLS interstitial ad breaks with client-side ad replacement, the emerging server-guided model.
- **A testable chain.** With the open source test ad server you can validate signalling, replacement, and tracking end to end before connecting a commercial ad stack.

The pattern is deliberate: the channel origination layer stays neutral and standards-based (HLS signalling), and the monetization layer remains a swappable choice.

COMBINING FAST WITH CLOUD-NATIVE PLAYOUT

FAST channel origination and broadcast playout used to be different worlds: one born on the web, one built from broadcast hardware. As playout becomes software-based and cloud-native, they converge into one stack, and the FAST components described here become the origination layer of a broader channel operation.

The key insight is that curation and scheduling can be shared while the output format is a deployment choice. The same schedule, whether written by editors, exchanged with broadcast planning systems (the stack includes BXF-based schedule exchange), or assembled by an AI curator, can drive two delivery heads. The FAST engine produces HLS for streaming platforms, FAST aggregators, and direct-to-app distribution. A software playout engine produces an MPEG-TS transport stream over SRT or UDP for media gateways, monitoring probes, and traditional distribution to TV operators. One channel definition, two outputs.

Graphics and branding fit the same file-based philosophy: channel graphics and menu boards can be produced with HTML5 graphics tooling and pre-rendered into ordinary clips that the engine plays like any other content, with heavier live branding applied in the playout head where a channel requires it. For live moments, VOD2Live channels can splice in genuine live HLS feeds, and the playout path handles the hard broadcast continuity cases. Eyevinn has built exactly this class of solution in production, including cloud-native software playout for linear channel distribution at a major broadcaster.

The practical consequence: a pop-up channel born as a FAST experiment can graduate into a fully distributed broadcast channel without being rebuilt. The curation layer, the scheduling layer, and the content supply chain stay the same; you add an output head.

DEPLOYMENT: MANAGED FIRST, NEVER LOCKED IN

Every component runs in three ways. As a **managed service in Open Source Cloud**, where a channel can be launched in minutes with no infrastructure and a free tier to start on. As **self-hosted containers** (Docker and docker-compose starting points are published) in your own cloud or data center. Or **embedded as a library** inside your own platform, implementing the asset and channel manager interfaces for full control.

Because the software is Apache 2.0 open source, the managed path and the self-hosted path are the same software. Start managed to prove the business case; if scale or procurement later favors running it yourself, you migrate the deployment, not the solution.

CONCLUSION

FAST channel creation does not require a broadcast playout budget. With VOD2Live as the engine principle, open source components as the stack, and managed services for instant launch, a content owner can put a linear channel on the air in an afternoon, program it with anything from a text file to an AI curator working from enriched metadata, and hand ad-ready streams to any SSAI vendor. The cost and energy profile makes long-tail and pop-up channels viable, and because the same stack extends into software-based cloud-native playout, the line between a FAST portfolio and a broadcast channel portfolio becomes a deployment choice rather than a technology boundary. The open source foundation means the solution is yours to keep.

Eyevinn Technology is an independent consultancy of video-streaming and broadcast software specialists in Stockholm, and the creator of the open source Channel Engine and the Open Source Cloud platform. We help content owners and distributors design, launch, and scale FAST channel operations, from first pilot channel to full portfolio with monetization. Independent means vendor-neutral: we help you select and integrate the right components, never sell you boxes.

FURTHER READING

- **Channel Engine (open source):** github.com/Eyevinn/channel-engine
- **FAST Channel Engine documentation:** fast.docs.eyevinn.technology
- **FAST channels with VOD2Live and open source components:** eyevinntechnology.medium.com
- **Dynamic FAST channels in Open Source Cloud:** eyevinntechnology.medium.com
- **Adding ad opportunities to FAST channels in Open Source Cloud:** eyevinntechnology.medium.com
- **Open Source Cloud:** osaas.io

Eyevinn Technology, independent video-streaming and broadcast specialists, Stockholm eyevinn.se · osaas.io