個人で 24 時間 TV 局を回す

放送自動化システム ICS-TV のアーキテクチャ

CasparCG × Django × gRPC で作る、止まらないリニア配信

八神 翔大 @shotayagami

WHO

自己紹介

八神 翔大

やがみ しょうた

X (旧Twitter)　**@shotayagami**

misskey　**@shotayagami@misskey.io**

よめくださいMAX 〜DIRTY MIX〜

経歴

- 港湾コンテナターミナルシステムの運営に 16 年。
- かたわら同人サークル運営で、自宅サーバ構築から Web 開発まで。
- その技術で 10 年前に Web システム開発へ転向。
- 〜2026年3月: 幼児教育教室を運営する会社で社内システム開発。
- **現在: 新規事業のために用意した会社の“雇われ経営者”。**

ORIGIN

開発のきっかけ ―― 「そういえば VTuber やってたなぁ」

- 2018〜2023年、なぜか VTuber 活動っぽいことをやっていた → YouTube チャンネルが既にあった。
- 2023年から会社勤務で時間がなくなり、3年ほど放置。
- せっかく収益化されていたのに、剥奪。
  - 半年放置すると収益化は消える…。
- でも過去資産は山ほどある ―― 活用しない手はない。

→ **「24時間365日 再放送する仕組みがあれば、復活できるんじゃない？」**

WHAT

番組表を編集すると、あとは人手ゼロで流れ続ける

- **ICS-TV は、個人で運用する 24 時間自動放送局。**
- YouTube Live へ 24/7 連続送出。番組・CM・テロップを自動で送り出す。
- **やることは「番組表に番組を置く」だけ。**
  - 録画番組も生中継も、CM 挿入も、フィラー(隙間埋め)も全部自動。
- 放送局のマスター運行に相当する監視・自動退避・障害復帰まで実装。
- **本番稼働中 (v0.7.0)。要件定義から約10日・個人開発・コードは ~36K 行。**

WHY

「OBS で手動配信」から「無人の放送局」へ

- 出発点: OBS を人が操作して YouTube に流す —— 人がいないと放送できない。
- **目標: ウェザーニュース LiVE / SOLiVE と同じ「リニア(線形)チャンネル」構造。**
  - 連続した1本の放送の上に、番組表・CM枠・2時間枠を論理的に重ねる。
  - YouTube = 無料公開、自社経路(Cloudflare) = 将来の有料、の二階建て。
- **放送は「止まったら事故」—— 24/7 で落ちない設計が全ての前提になる。**

PART 1

全体アーキテクチャ

制御プレーン / 送出ノード / クラウド —— 役割で割った3層

ARCHITECTURE

役割で割った3層 —— 疎結合で繋ぐ

制御プレーン
自宅 Proxmox / RKE2

- Django + PostgreSQL
- 番組編成 UI・番組表公開
- スケジューラ(Celery)
- as-run を解決して push
- 営放 / 納品 / 課金 / 計測

送出ノード
自宅 LXC (CT131)

- playout agent (Python)
- CasparCG (Linux/GPU)
- MediaMTX (SRT/RTMP終端)
- ローカル SSD キャッシュ
- 数十時間分を自律実行

クラウド配信
Cloudflare / YouTube

- CF Live Input → Output
- YouTube 永続キーへ連続送出
- Data API で 2h×12 rolling
- HLS/DASH (将来サブスク)
- 素材は R2 (egress無料)

gRPC で as-run を push ／ AMCP で CasparCG を駆動 ／ RTMP・SRT で生入力 ／ HLS で配信

STACK

主要コンポーネントと技術

コンポーネント	技術	役割
制御プレーン	Django + PostgreSQL + Celery	編成・スケジューラ・業務ロジック・公開Web
送出 agent	Python (asyncio) + gRPC + SQLite	as-run をローカル実行・store-and-forward
送出エンジン	CasparCG (Linux/GPU) + AMCP	映像合成・テロップ・NVENC エンコード
生入力	MediaMTX + Cloudflare One/WARP	現場 SRT を公開IPなしでローカル終端
素材	OMV → FFmpeg正規化 → R2	解像度/fps/コーデック/-14LUFS を統一
配信	Cloudflare Live + YouTube Data API	永続キーへ連続送出・2h枠 rolling
デプロイ	RKE2 + ArgoCD (GitOps) + Helm	制御プレーンを K8s で git 駆動デプロイ

KEY DESIGN

設計の背骨 —— 「編成」と「送出」を分ける

- **編成 (人が組む): 番組を任意時刻に置くタイムライン。自由で、いつでも書き換わる。**
- **送出 (機械が実行する as-run): フレーム単位の実行命令列。決定的で、不変。**

- 分離の効能: 営放(広告)も納品も resolver へ依存逆流しない。送出層は編成を知らない。
- 業務系が落ちても放送は止まらない —— 切り離せるから 24/7 が成立する。

YouTube 上でリニア ch を回す

途切れたら終わり、を逆手に取る

DEEP DIVE ①

リニアの逆説

- YouTube Live / CF Live Input は、フレームが途切れない連続入力を要求する。
  – 止めたら配信が壊れる。番組ごとにストリームを張り直す、は不可能。
- **発想の転換:**
- **チャンネルごとに「常時稼働の送出を1本」持ち、その上を番組表で切り替える。**
  – これは放送局の linear playout そのもの。番組・CM・2h枠は連続ストリーム上の論理的な区切りにすぎない。
- **CasparCG を永遠に回し続け、約30秒〜分単位で「上に載るもの」だけを差し替える。**

DEEP DIVE ①

編成 → as-run の解決（リゾルバ）

人が置いた **編成** を、リゾルバが **as-run**（不変な実行命令列）へ解決する。

① 編成 — 人が任意時刻に番組を置く

② as-run — [now, +48h] を冪等に解決

```
def resolve(channel, t0, t1):    # t1 = now+48h
    cursor = t0
    for p in programs_in(channel, t0, t1):
        if p.start_at > cursor:          # 隙間
            events += emit_filler(cursor, p.start_at)
        events += (emit_recorded(p) if p.recorded
                   else emit_live(p))
        cursor = p.end_at
    upsert_playout_events(events)  # ikey で diff
    push_to_agent(channel, events)
```

隙間→フィラー / CM枠→15·20sグリッド自動充填 / 生→cutのみ事前生成。重なりは `EXCLUDE USING gist` がDBで拒否、実行済みは不変・未来分のみ差替。

DEEP DIVE ①

継ぎ目のない切替 (LOADBG → PLAY)

- agent は scheduled_at の数秒前に背面ロード、定刻でテイク —— これでフレーム精度カット。

① LOADBG

次クリップを
裏レイヤへ先読み

PREROLL ≈ 5s 前

② PLAY

定刻 (NTP同期) に
テイク=切替

TICK = 100ms

③ filler loop

次イベントの
until まで再生

黒画面を作らない

- 素材は事前に正規化 (解像度/fps/コーデック/音声 -14 LUFS) —— 継ぎ目で映像も音量も乱れない。
- CG (Lバー/速報/提供) は別レイヤに常駐。背面が差し替わってもテロップは出っぱなし。

DEEP DIVE ①

YouTube への送出 —— 連続性と枠分割を分離

- CF Live Output が YouTube の永続ストリームキーへ「連続送出」。送出は一度も切れない。
- その上で Data API が 2時間ごとに liveBroadcast を rolling 生成し、
  - testing → live → complete を順送りでトランジション。
- **物理的な送出の連続性と、2h という枠分割は、独立に成立する。**

▸ 送出は CF へ1系統だけ → CF からファンアウトで自宅の上り帯域が半分。

実機: チャンネル設定 (ch2)

永続 liveStream + 2h × 12 rolling (Celery: 10分生成 / 1分 transition) + Cloudflare Live。

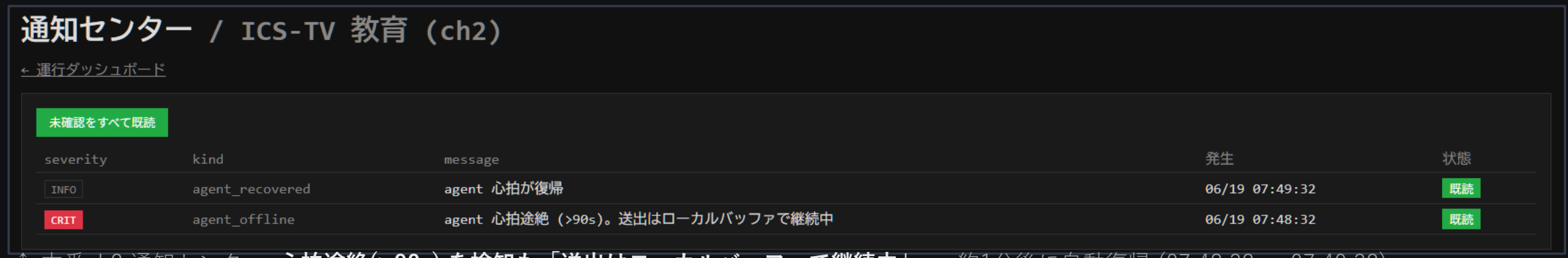
WAN 断でも止まらない agent

制御が落ちても、放送は続く

DEEP DIVE ②

自宅集約という制約 → ローカル自律

- 制御プレーンも送出ノードも自宅。両者を繋ぐ WAN は、いつか必ず切れる。
- **agent が as-run を「数十時間分」ローカル SQLite に保持 —— 実機では queue = 693 件 (seq 62026)**。
  - 制御プレーンが落ちても、agent はローカルキューで放送を続行 (隙間はフィラー)。実績は store-and-forward で WAN 復帰時に返送。
- 広告の放送回数も「実送出確定時」に確定 —— 欠送を過大計上しない。



↑ 本番ch2 通知センター: **心拍途絶(>90s) を検知も「送出はローカルバッファで継続中」** → 約1分後に自動復帰 (07:48:32 → 07:49:32)。

DEEP DIVE ②

冪等な as-run —— 再起動・再送に強い

冪等キーは (channel・時刻・action・seg) から決定的に算出:

```
ikey = uuid5(NS, f"{ch}:{scheduled_at}:{action}:{seg}")
```

- 再解決しても同じキー → upsert が「自然な差分」。実行済/実行中は不変、未来分のみ差替。
- agent は実行済キーを SQLite に永続化 → 再起動・WAN復帰後は再 pull で実行済をスキップ。
- 削除は tombstone (CANCELLED) として配信 —— カスケード失敗を作らない。
- **結果: 二重送出しない・取りこぼさない。実行層をほぼステートレスに保てる。**

実機: AS-RUN ライブログ

AS-RUN ライブログ（新しい順）

時刻	action	status	番組	note
22:13:12	play_filler	scheduled	–	
22:08:12	play_filler	cancelled	–	
22:07:26	play_filler	scheduled	–	
22:03:12	play_filler	cancelled	–	
22:02:26	play_filler	cancelled	–	
22:01:10	play_filler	scheduled	–	
21:58:12	play_filler	cancelled	–	
21:57:26	play_filler	cancelled	–	
21:56:10	play_filler	cancelled	–	
21:56:00	play_filler	scheduled	–	
21:53:12	play_filler	cancelled	–	
21:52:26	play_filler	cancelled	–	
21:51:10	play_filler	cancelled	–	
21:51:00	play_filler	cancelled	–	
21:50:52	play_filler	scheduled	–	
21:48:12	play_filler	cancelled	–	
21:47:26	play_filler	cancelled	–	
21:46:10	play_filler	cancelled	–	
21:46:00	play_filler	cancelled	–	
21:45:52	play_filler	cancelled	–	

scheduled と cancelled(=tombstone) が冪等キーで差分反映。

DEEP DIVE ②

3層ウォッチドッグ + 自動退避・復帰

ウォッチドッグは3層で、重複なく分担する:

- **ホスト層** watchdog.sh (systemd 30s) — プロセス死活の最後の砦
- **agent層** feed monitor (asyncio) — feed断を2秒で検知 → 自動スレート退避
- **server層** 死活 beat (Celery) — heartbeat 途絶 90s で通知

▸ 生フィード断 → 2秒で「しばらくお待ちください」へ自動退避。復帰は誤爆防止のヒステリシス(10s連続)で本線へ戻し、フラップ制限(10分3回)も持つ。

▸ ハマった盲点「AMCP健全 ≠ 出力健全」—— 出力経路にも watchdog を足した。

実機: 運行ヘルス (ch2)

STATUS

規模と現状

v0.7.0
本番稼働中
(YouTube ライブ)

~36K 行
サーバ31K + agent4.6K
+ frontend / proto

410
コミット
約10日 / 個人開発

2 ch
全機能を検証後
最大4chへ複製予定

- 稼働: 自宅 Proxmox の RKE2 (制御) + LXC (送出) + Cloudflare/YouTube。Zabbix で監視。
- 実装済: 録画/生/CM・正規化パイプライン・営放(広告)・納品QC・課金・視聴計測・WSライブコメント・権利管理。
- 次の一手: マルチch複製・サブスク視聴ゲート・送出ノード冗長化・React/API 化。

BREADTH

放送の周りも全部 —— 周辺サブシステムまで個人で

- **放送を成立させる“裏方”も、同じ1システムに:**

- 納品QCポータル — 外部制作会社が入稿、技術+R128ラウドネス+リーダー検出を自動QC、回(Episode)単位で確定。

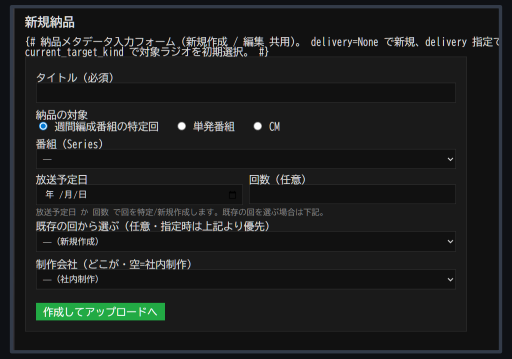
- 営放(広告) — タイム/スポット契約・競合排他・放確台帳。

- 請求 — 放送確認書/請求書を内製PDF、番組予算(AP)も管理。

- 視聴計測 / ライブコメント(WS) / 会員 / サブスク課金 / 権利管理。

▸ どれも送出系と疎結合 —— 落ちても放送は止まらない。

実機: 納品QCポータル



← 納品マトリクス
本編=approved が多数

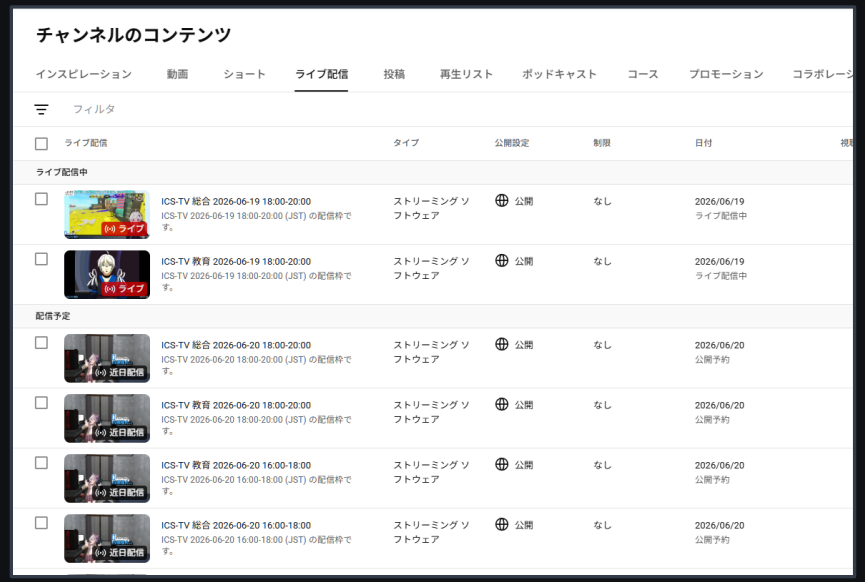
← 新規納品フォーム
回(Episode)を指定して入稿

REAL DATA

24時間配信は、実際に回って・見られている

321	17.3h	1,183	203
視聴回数 / 週	総再生時間 / 週	登録者（累計）	視聴回数 / 48h

直近7日のアナリティクス（無人運用）。視聴が右肩上がりで積み上がる。



「総合」「教育」の2chが同時ライブ＋翌日枠を**自動「配信予定」**生成＝2h×12 rolling×マルチchが稼働中。

TAKEAWAYS

持ち帰ってほしい5つ

- リニアの逆説: 「常時1本の送出」を持てば、放送局型の自動編成は個人でも作れる。
- 編成と送出の分離: 人が書く編成と、機械が実行する不変な as-run を分ける。業務系を切り離せる
- 冪等 as-run + store-and-forward: 決定的キーで実行層をほぼステートレスに。クラッシュにもWAN断にも強い。
- ローカル自律 + 多層監視: 制御が落ちても放送は続く。これが 24/7 を成立させる。

- よめください

ありがとうございました

個人で 24 時間 TV 局を回す —— ICS-TV

CasparCG × Django × gRPC

Questions?