Sapsan Manual

Servidor de mídia de streaming

Flussonic

Table of contents

1. Produtos

2. Sapsan

2.0.1 Sapsan

O Flussonic Sapsan (a seguir, Sapsan) é um servidor de mídia de streaming. Ele captura vídeo ao vivo por protocolos padrão da indústria, processa-o (transcodificação, montagem multibitrate, capturas), grava-o em seu próprio arquivo DVR em disco e o entrega aos espectadores tanto ao vivo quanto a partir do arquivo.

Propósito: a captura, o processamento, a gravação e a entrega de vídeo ao vivo em projetos de streaming pela internet, IPTV e videomonitoramento. A lista detalhada de funções está na página características funcionais; o desenho do sistema, na página arquitetura técnica.

O Sapsan é controlado por um único arquivo de configuração, `sapsan.yaml`, que é recarregado em tempo real sem interromper o streaming, e pode funcionar tanto de forma autônoma quanto sob o controle de um sistema de gestão externo (Flussonic Central).

Funcionalidades principais

- **Captura**: RTSP (câmeras IP), MPEG-TS (UDP e HTTP), SRT, RTMP, um canal a partir de um arquivo, fonte sintética de teste.
- **Publicação**: RTMP publish, SRT publish, WebRTC WHIP.
- **Reserva da captura**: várias fontes por stream com comutação automática (LSI) e um arquivo de reserva.
- **Transcodificação**: preparação da escada de qualidades multibitrate (MBR) baseada em FFmpeg — h264/hevc/av1, aac/opus.
- **Reprodução**: HLS e LL-HLS (fMP4), DASH, MPEG-TS por HTTP (incluindo CBR), SRT, RTSP, WebRTC WHEP.
- **Retransmissão**: push de um stream por RTMP, SRT e UDP (multicast/unicast).
- **DVR**: arquivo append-only em vários discos, reprodução do arquivo, rewind, prévias de quadros, reprodução sem interrupções de arquivos de outros servidores (remotes).
- **Capturas**: prévias JPEG dos streams ao vivo e do arquivo.
- **Autorização**: tokens de reprodução e publicação, backends de autorização externos.
- **ONVIF**: descoberta de câmeras na rede, eventos de movimento e detecção.
- **Gestão**: Admin API, configuração externa (`config_external`), recarga a quente por SIGHUP, compatibilidade com as URLs e a API do Flussonic.
- **Gateway Peeklio (rproxy)**: acesso a câmeras e dispositivos atrás de NAT por meio de agentes.
- **Observabilidade**: logs estruturados, métricas do Prometheus, traces de OpenTelemetry, contadores de qualidade da captura.

Início rápido

- Instalar e executar o Sapsan
- Configurar seu primeiro stream
- Reproduzir o stream por HLS

Navegador da documentação

Esta documentação vai ajudar você a:

- Instalar e executar o Sapsan
- Entender a configuração
- Capturar vídeo:
  - de uma câmera IP por RTSP
  - de MPEG-TS em multicast ou por HTTP
  - por SRT
  - por RTMP
  - criar um canal a partir de um arquivo
  - executar um stream de teste
  - aceitar publicações RTMP/SRT/WHIP
- Configurar fontes reserva
- Montar um stream multibitrate a partir de várias fontes
- Descobrir câmeras ONVIF e receber seus eventos
- Transcodificar um stream
- Entregar vídeo aos espectadores:
  - HLS e LL-HLS
  - DASH
  - MPEG-TS
  - SRT
  - RTSP
  - WebRTC
- Retransmitir a outros servidores
- Gravar o arquivo e reproduzi-lo
- Reproduzir sem interrupções um arquivo de outro servidor
- Obter capturas do stream
- Proteger a reprodução e a publicação
- Operar o servidor:
  - Admin API
  - Gestão externa da configuração
  - Gateway Peeklio e agentes
  - Monitoramento, logs e traces
  - Licenciamento
  - Compatibilidade com o Flussonic

2.1 Características funcionais

A lista completa de funções do servidor de mídia de streaming Flussonic Sapsan. Cada função está descrita no guia do administrador (links no texto).

2.1.1 Captura de vídeo

- Captura de câmeras IP e servidores de mídia por RTSP (RTP/AVP/TCP interleaved) com autenticação Basic e Digest — detalhes. Codecs: vídeo H264, HEVC, VP8, MJPEG; áudio AAC, G.711, Opus, MPEG-4; metadados ONVIF.
- Captura de MPEG-TS de UDP (unicast, multicast IPv4/IPv6) e por HTTP — detalhes. Codecs: vídeo H264, HEVC, MPEG-2; áudio AAC, AC3, EAC3, MPEG-2; teletexto, KLV, SCTE-35.
- Captura por SRT em modo listener com criptografia AES e controle de streamid — detalhes.
- Pull por RTMP, incluindo autenticação Adobe/FMS e enhanced RTMP (HEVC) — detalhes.
- Um canal a partir de um arquivo MP4 local (MBR multipista) — detalhes.
- Um gerador de sinal sintético de teste integrado, com um timecode legível por máquina na imagem — detalhes.
- Publicação: RTMP publish, SRT publish, WebRTC WHIP; proteção contra sequestro do ar (um publicador por stream) — detalhes.
- Reserva de fontes com comutação automática sem tela preta (LSI), um arquivo de reserva e controle das entradas por meio de arquivos de bandeira externos — detalhes.
- Montagem de um stream multibitrate a partir de várias fontes — detalhes.
- Descoberta de câmeras ONVIF, captura de eventos de movimento e detecção, verificação de compatibilidade das câmeras — detalhes.

2.1.2 Processamento de vídeo

- Transcodificação baseada em FFmpeg: vídeo H264/HEVC/AV1, áudio AAC/Opus, escada de qualidades multibitrate com quadros-chave alinhados, redimensionamento (scale/fit/crop), controle do GOP — detalhes.
- Capturas JPEG periódicas de um stream (da faixa de vídeo ou de uma URL de snapshot da câmera), guardadas no arquivo — detalhes.

2.1.3 Entrega de vídeo

- HLS e LL-HLS (fMP4/CMAF, protocolo v9): segmentos parciais, blocking reload, atualizações delta das playlists — detalhes.
- MPEG-DASH (ao vivo e arquivo), multibitrate em um único AdaptationSet — detalhes.
- MPEG-TS por HTTP, incluindo CBR estrito com PCR corretos e tratamento do HRD — detalhes.
- Saída SRT com criptografia — detalhes.
- Um servidor RTSP (H264/HEVC/AAC) — detalhes.
- WebRTC WHEP com latência mínima — detalhes.
- Retransmissão: push por RTMP (incluindo HEVC/AV1), SRT e UDP multicast — detalhes.

2.1.4 Gravação e arquivo (DVR)

- Gravação em seu próprio armazenamento append-only distribuído por vários discos, com balanceamento automático — detalhes.
- Políticas de retenção por profundidade e tamanho, episódios protegidos, proteção automática contra estouro dos discos.
- Autorreparação do catálogo do arquivo após falhas e mudanças de discos.
- Reprodução do arquivo por HLS/DASH: rewind com emenda sem interrupções com o sinal ao vivo, acesso por tempo absoluto — detalhes.
- Reprodução sem interrupções de arquivos gravados em outros servidores, com replicação preguiçosa — detalhes.

2.1.5 Gestão e segurança

- Configuração em um único arquivo YAML, com validação e recarga a quente sem interromper o streaming — detalhes.

- Gestão da configuração a partir de um servidor externo (Flussonic Central) com aplicação à prova de falhas — detalhes.
- Autorização de reprodução e publicação: tokens estáticos, backends HTTP externos, registro de sessões — detalhes.
- Uma Admin API (compatível com a Flussonic API v3, mais a v4 nativa) — detalhes.
- Compatibilidade com os esquemas de URL do Flussonic Media Server — detalhes.
- O gateway Peeklio para alcançar câmeras atrás de NAT por meio de agentes — detalhes.
- Licenciamento com arquivos de licença assinados — detalhes.

2.1.6 Observabilidade

- Logs estruturados, métricas do Prometheus, traces de OpenTelemetry — detalhes.
- Contadores de qualidade da captura (MPEG-TS, SRT); estatísticas de fontes, sessões e DVR.
- Validadores de HLS e DASH integrados para o diagnóstico da entrega.

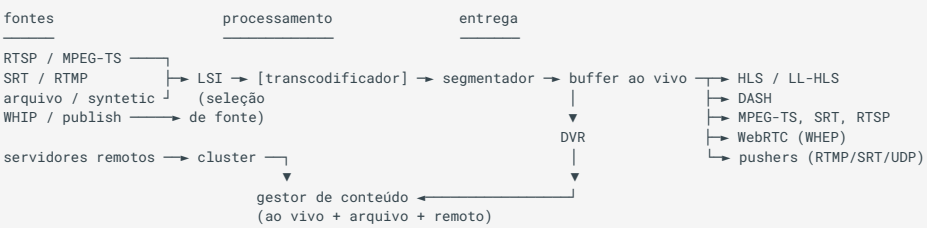
2.2 Arquitetura técnica

O Flussonic Sapsan é uma aplicação de servidor escrita em Rust. Todos os subsistemas rodam dentro de um único processo `sapsan` como tarefas do runtime assíncrono tokio e interagem por canais de mensagens tipados — sem estado mutável compartilhado entre subsistemas.

2.2.1 Componentes

Componente	Função
Gestor	carregamento e validação da configuração, partida dos subsistemas, vinculação dos listeners de rede, recarga a quente, Admin API
Listeners de rede	HTTP (HLS, DASH, MPEG-TS, WHIP/WHEP, API), RTSP, RTMP, WebRTC (UDP), portas SRT por stream
Gestor de streams	ciclo de vida dos streams: entradas e comutação de fontes (LSI), publicação, pushers, capturas
Transcodificador	decodificação e codificação (FFmpeg): H264/HEVC/AV1, AAC/Opus, a escada de qualidades multibitrate
Segmentador e buffer ao vivo	fatiamento do stream em fragmentos fMP4, o buffer circular da borda ao vivo
Gestor de conteúdo	a superfície unificada de conteúdo: reúne o buffer ao vivo, o arquivo local e os arquivos remotos; todos os protocolos de entrega leem dela
DVR	armazenamento do arquivo em disco: blobs append-only por hora, um catálogo sobre um banco de dados chave-valor embutido, um indexador em segundo plano (reconstrução, preenchimento de metadados, limpeza)
Cluster	leitura de arquivos de servidores Sapsan remotos pelo streaming-api interno
Sessões e autorização	registro de sessões de espectadores e publicadores, tokens, backends de autorização externos
Gateway Peeklio	um proxy reverso para dispositivos atrás de NAT: registro de agentes, túneis
Telemetria	logs estruturados, métricas do Prometheus, traces de OpenTelemetry

2.2.2 Fluxos de dados



- **Caminho de escrita**: a entrada ativa do stream (a escolhida pela LSI) entrega os quadros; se for preciso, eles passam pelo transcodificador; o segmentador monta fragmentos fMP4, que entram no buffer ao vivo e, em paralelo, são anexados ao DVR.
- **Caminho de leitura**: um módulo de protocolo pede os dados ao gestor de conteúdo, que escolhe a fonte de forma transparente — o buffer ao vivo, o arquivo local ou um servidor remoto. Os reprodutores nunca sabem de onde os dados vieram fisicamente.
- **Isolamento de leitura e escrita no DVR**: a escrita de cada stream é serializada na sua própria tarefa; as leituras a contornam e vão direto aos blobs por meio de um cache compartilhado — os leitores não disputam com o escritor.

2.2.3 Endereçamento de conteúdo

A unidade de armazenamento e entrega é o fragmento, endereçado por tempo UTC absoluto (DTS + timescale). O mesmo nome de fragmento vale no buffer ao vivo, no arquivo e em todos os servidores do cluster. Essa propriedade sustenta a emenda sem interrupções do ao vivo com o arquivo e a fusão de arquivos entre servidores. O contrato de URL (rotas, URIs das playlists, referências a fragmentos) está definido em um único módulo streaming-api, comum ao servidor e aos clientes do cluster.

2.2.4 Ciclo de vida da configuração

1. A configuração é lida de `sapsan.yaml` e validada como um todo; uma configuração inválida não é aplicada.
2. Ao receber SIGHUP, a configuração é relida: os streams, a autorização e o DVR são reconfigurados no lugar; os listeners são revinculados somente quando uma porta muda.
3. Com `config_external` ativado, a lista de streams é buscada periodicamente de um servidor externo; as atualizações parcialmente inválidas são aplicadas sem os streams quebrados, e as totalmente inválidas são descartadas mantendo a configuração em funcionamento.

2.2.5 Pilha tecnológica

Camada	Tecnologia
Linguagem	Rust
Runtime assíncrono	tokio (E/S de rede, tarefas), rayon (paralelismo de CPU)
Alocador	jemalloc
Transcodificação	FFmpeg
Catálogo do DVR	banco de dados chave-valor embutido (LSM)
Observabilidade	OpenTelemetry (OTLP), Prometheus

3. Administrador

3.1 Primeiros passos

3.1.1 Instalação e execução do Sapsan

Esta página descreve como instalar o Sapsan, ativá-lo com uma licença, executá-lo com uma configuração mínima e verificar que o servidor funciona.

Antes da instalação

O Sapsan não funciona sem licença: o arquivo `license.txt` precisa existir antes do primeiro início, e o servidor precisa de acesso à internet para a ativação online. Consulte o licenciamento para conhecer os detalhes.

> **Note**
>
> TODO: requisitos de sistema (SO, arquiteturas, CPU/RAM, discos para DVR).

Instalação a partir do pacote deb

O Sapsan é distribuído como pacote deb `sapsan` do repositório da Flussonic:

```
curl -sSf http://apt.flussonic.com/binary/gpg.key > /etc/apt/trusted.gpg.d/flussonic.gpg
echo "deb http://apt.flussonic.com binary/" > /etc/apt/sources.list.d/flussonic.list
apt update
apt install sapsan
```

O pacote instala o binário `/usr/sbin/sapsan`, a unidade de systemd `sapsan.service`, a configuração padrão em `/etc/sapsan/sapsan.yaml` e cria o diretório de estado `/var/lib/sapsan`.

Ativação

Antes de iniciar, coloque o arquivo de licença recebido junto à configuração:

```
cp license.txt /etc/sapsan/license.txt
chmod 600 /etc/sapsan/license.txt
```

No primeiro início, o Sapsan se ativa online e grava os artefatos de ativação no diretório de estado `/var/lib/sapsan`:

Arquivo	Conteúdo
`server.id`	identificador único do servidor (UUID)
`activation-<version>.json`	ativação para uma versão específica do produto
`keys/`	diretório com o material de chaves gerado pelo Sapsan

Consulte o licenciamento para ver os estados da licença e como verificá-los.

Configuração mínima

O Sapsan lê sua configuração de `sapsan.yaml` (o caminho pode ser redefinido com a variável de ambiente `CONFIG_PATH`). Uma configuração mínima é uma porta HTTP e um stream:

```
listeners:
  http:
    - port: 80
```

```
streams:
  - name: demo
    inputs:
    - syntetic: {}
    transcoder:
      output:
      - source: !content video
        codec: !set h264
      - source: !content audio
        codec: !set aac
```

`streams` é uma lista: cada stream é um item com seu próprio `name`. Para avançar ao vídeo real, consulte a captura de vídeo.

Execução no Docker

A imagem oficial é `flussonic/sapsan`. Mapeie as portas e monte a configuração, a licença e o diretório de estado:

```
mkdir -p config data
# coloque config/sapsan.yaml e config/license.txt no lugar

docker run --rm -p 80:80 -p 554:554 -p 1935:1935 \
  -v "$(pwd)/config/sapsan.yaml:/etc/sapsan/sapsan.yaml:ro" \
  -v "$(pwd)/config/license.txt:/etc/sapsan/license.txt:ro" \
  -v "$(pwd)/data:/var/lib/sapsan" \
  flussonic/sapsan:latest
```

O diretório `data` (`/var/lib/sapsan`) precisa ser gravável e persistir entre reinícios — nele vivem `server.id`, os arquivos de ativação e o diretório `keys/`. Os caminhos `CONFIG_PATH` e `SERVER_ID_PATH` já estão definidos na imagem.

Início e parada

Quando instalado a partir do pacote deb, o servidor é gerenciado pelo systemd:

```
systemctl start sapsan     # iniciar
systemctl status sapsan    # estado
systemctl stop sapsan      # parar
systemctl enable sapsan    # iniciar na inicialização do sistema
```

Recarregue a configuração sem interromper o streaming com um sinal:

```
kill -HUP "$(systemctl show -p MainPID --value sapsan)"
```

Verificação de funcionamento

Verifique se o processo está rodando:

```
systemctl status sapsan
```

Verifique se a licença está ativada:

```
curl http://localhost/streamer/api-v4/license/status
```

O estado `Licensed` significa que o Sapsan está instalado, ativado e pronto. Abra também a playlist do stream de teste:

```
curl http://localhost/streaming/v/demo/index.m3u8
```

Se o servidor devolver uma playlist, o stream de demonstração funciona. Em seguida, configure a captura de vídeo real.

3.1.2 Configuração do Sapsan

Toda a configuração do Sapsan vive em um único arquivo YAML, `sapsan.yaml`. Esta página descreve suas seções e as regras de recarga.

Estrutura do arquivo

```
listeners:        # portas em que o servidor escuta
  http:
    - port: 80
  rtsp:
    - port: 554
  rtmp:
    - port: 1935
  webrtc:
    - port: 5005

streams:          # lista de streams
  - name: cam1
    inputs:
    - rtsp:
        url: rtsp://admin:password@10.0.0.5/stream0
    dvr: {}

dvr:              # armazenamento compartilhado do arquivo
  root: /storage
  disks:
  - path: d1

auth: {}            # autorização de reprodução e publicação
config_external: {} # gestão externa da configuração
rproxy: {}          # gateway Peeklio
api_auth:           # login/senha do Admin API
  login: admin
  password: secret
```

Seção	O que configura	Detalhes
`listeners`	portas HTTP, RTSP, RTMP e WebRTC do servidor	esta página
`streams`	streams: fontes, transcodificador, DVR, push, capturas de tela	captura
`dvr`	armazenamento em disco do arquivo	gravação DVR
`auth`	tokens e backends de autorização	autorização
`config_external`	obtenção da configuração de um servidor externo	configuração externa
`rproxy`	gateway Peeklio para agentes	gateway Peeklio
`api_auth`	acesso ao Admin API	Admin API

Validação

O Sapsan valida a configuração no carregamento e se recusa a iniciar com uma inválida:

- campos desconhecidos são erro (proteção contra erros de digitação);
- cada entrada de um stream deve conter exatamente um protocolo;
- as portas não podem ser reutilizadas entre listeners e streams;
- se um stream tem `dvr` habilitado, a seção global `dvr` deve existir.

Recarga a quente

Ao receber `SIGHUP`, o Sapsan relê o arquivo `sapsan.yaml` e aplica as mudanças sem reiniciar o processo:

- streams, autorização e DVR são reconfigurados na hora;
- os listeners só voltam a se vincular quando uma porta muda;
- o gateway rproxy mantém as conexões dos agentes entre recargas (mas trocar `streampoint_key` / `endpoint_auth_url` exige reiniciar o processo).

```
kill -HUP $(pidof sapsan)
```

3.2 Captura

3.2.1 Captura por RTSP

Sapsan puxa o vídeo de câmeras IP e de outras fontes RTSP em modo cliente (pull).

Configuração

```
streams:
  - name: cam1
    inputs:
    - rtsp:
        url: rtsp://admin:password@10.0.0.5:554/stream0
        peer_timeout_ms: 5000
```

Parâmetro	Descrição
`url`	endereço da câmera; login e senha vão na URL
`peer_timeout_ms`	tempo limite de inatividade da fonte, após o qual a entrada é considerada caída
`via_agent`	captura por um agente Peeklio (`agent://ID`) quando a câmera está atrás de um NAT — veja o gateway Peeklio

Autenticação

Basic e Digest são suportados (com reversão automática para Basic quando a câmera recusa Digest). As credenciais vêm da URL e não são enviadas em claro para a câmera quando se usa Digest.

Codecs e transporte

O transporte é RTP/AVP/TCP (interleaved), robusto diante de NAT e firewalls. São decodificados: vídeo H264, HEVC, VP8, MJPEG; áudio AAC, G.711 (PCMA/PCMU), Opus, MPEG-4; metadados ONVIF (eventos da câmera) direto do stream RTP. O transbordamento e o recuo dos timestamps RTP são tratados.

O parser SDP foi testado a fundo com câmeras reais: Axis, Hikvision, Dahua, Bosch, Panasonic, Uniview, Beward e outras — incluindo câmeras com SDP malformado (PPS ausente, SEI quebrado).

> **Note**
>
> O áudio G.711 não toca por HLS — adicione um transcodificador que recodifique o áudio para AAC.

Aplicação das mudanças

A troca de `url` ou `via_agent` reinicia a fonte; a troca apenas de `peer_timeout_ms` é aplicada em tempo real.

Verificação

Abra `http://server/streaming/v/cam1/index.m3u8` em um reprodutor ou peça uma captura de tela em `http://server/streaming/live-preview-jpeg/cam1` .

O que vem a seguir

- Adicione uma fonte reserva
- Combine duas qualidades da câmera em um stream MBR
- Descubra câmeras por ONVIF
- Grave o stream no arquivo

3.2.2 Captura de MPEG-TS

Sapsan captura streams MPEG-TS SPTS de duas maneiras: escutando UDP (unicast e multicast, IPv4 e IPv6) e puxando o stream por HTTP.

Captura por UDP

```
streams:
  - name: tv1
    inputs:
    - udp:
        host: 239.0.0.1
        port: 5000
        peer_timeout_ms: 5000
```

Se `host` for um grupo multicast, o Sapsan se inscreve nele por conta própria (IGMP/MLD). `peer_timeout_ms` define o tempo limite de silêncio, após o qual a entrada é considerada caída e o LSI comuta para a reserva.

Captura por HTTP

```
streams:
  - name: tv2
    inputs:
    - tshttp:
        url: http://origin.example.com/tv2/mpegts
        peer_timeout_ms: 5000
```

A transferência chunked é suportada. Uma conexão rompida e uma fonte silenciosa são distinguidas (`PeerClosed` / `PeerTimeout`) e aparecem no status da entrada.

O que é decodificado

Vídeo H264, HEVC, MPEG-2; áudio AAC, AC3, EAC3, MPEG-2; teletexto DVB, metadados KLV, marcadores SCTE-35; descritores de idioma das trilhas e descritores de legendas DVB do PMT.

Aplicação das mudanças

No recarregamento da configuração, a troca de `host` / `port` (UDP) ou `url` (HTTP) reinicia a fonte; a troca apenas de `peer_timeout_ms` é aplicada em tempo real, sem interromper a captura.

Contadores de qualidade

São coletados contadores de qualidade de recepção por PID: pacotes e quadros, erros CC (violações do continuity counter, inclusive através dos limites de intervalos), erros TEI, pacotes scrambled, erros de CRC no PSI, PES quebrados, problemas do buffer do decodificador (HRD). Consulte-os na Admin API e no monitoramento.

O que vem a seguir

- Configure uma fonte reserva
- Devolva o stream para a rede como SPTS
- Faça push do stream para multicast

3.2.3 Captura por SRT

Sapsan captura um stream SRT escutando em uma porta UDP dedicada (modo listener). Cada stream recebe a sua própria porta.

Configuração

```
streams:
  - name: event
    inputs:
    - srt:
        host: 0.0.0.0
        port: 9000
        passphrase: verysecretpass
```

Parâmetro	Descrição
`host`, `port`	endereço e porta UDP em que se aguarda a conexão SRT
`passphrase`	chave de criptografia (AES); chave divergente faz a conexão ser recusada
`stream_id`	streamid esperado do cliente; a forma estilo Flussonic `#!::r=<nome>` é repassada como está
`handshake_timeout_ms`, `peer_timeout_ms`	tempos limite de handshake e de inatividade

Você pode enviar um stream para o Sapsan assim:

```
ffmpeg -re -i input.mp4 -c copy -f mpegts \
  "srt://server:9000?passphrase=verysecretpass"
```

Confiabilidade da entrega

O mecanismo ARQ completo do SRT está implementado: confirmações (ACK/ACKACK), pedidos de retransmissão dos pacotes perdidos (NAK), entrega em ordem a partir do buffer de latência, descarte dos pacotes irremediavelmente atrasados (too-late drop) e keepalive para conexões ociosas. O período do NAK acompanha o RTT medido.

Aplicação das mudanças

A troca de endereço, porta, `passphrase` ou `stream_id` reinicia a fonte; a troca apenas dos tempos limite é aplicada em tempo real.

Contadores

Por conexão: pacotes e bytes recebidos, RTT e a sua variância, pacotes perdidos e retransmitidos nos dois sentidos, tamanhos de buffer, tempos limite de keepalive. Consulte na Admin API.

O que vem a seguir

- Sirva o stream por SRT aos espectadores
- Retransmita o stream por SRT para outro servidor

3.2.4 Captura por RTMP

Sapsan pode puxar um stream RTMP de outro servidor em modo cliente (pull). Para aceitar publicações RTMP de entrada (push para o Sapsan), veja publicação.

Configuração

```
streams:
  - name: relay
    inputs:
    - rtmp:
        url: rtmp://origin.example.com/live/streamkey
        peer_timeout_ms: 5000
```

Se o servidor exigir autenticação, coloque o login e a senha na URL: `rtmp://user:password@origin.example.com/live/streamkey` — a autenticação em várias etapas da Adobe/FMS (`authmod=adobe`) é suportada.

Codecs

H264 + AAC, HEVC (enhanced RTMP), streams só de áudio. Metadados quebrados (`onMetaData`) não interrompem a captura.

Comportamento diante de erros

Os códigos de recusa do servidor são distinguidos e aparecem no status da entrada: stream não encontrado, acesso negado, stream parado, erro de protocolo. Uma conexão rompida e um servidor silencioso (`peer_timeout_ms`) também são distinguidos. O LSI comuta para uma entrada reserva em qualquer desses casos.

Aplicação das mudanças

A troca de `url` reinicia a fonte; a troca apenas de `peer_timeout_ms` é aplicada em tempo real.

O que vem a seguir

- Configure uma fonte reserva
- Retransmita o stream adiante por RTMP

3.2.5 Captura por HLS

O Sapsan segue uma playlist HLS de terceiros em modo cliente (pull): sonda a playlist, baixa os segmentos e entrega quadros ao pipeline tal como RTSP, SRT ou MPEG-TS. É assim que entrega o stream quase todo agregador, toda CDN de terceiros e todo parceiro a quem não foi permitido abrir UDP ou SRT.

Configuração

```
streams:
  - name: chan1
    inputs:
    - hls:
        url: https://cdn.example.com/live/index.m3u8
```

Parâmetro	Por padrão	Descrição
`url`	—	endereço da playlist exatamente como o operador o escreveu
`headers`	nenhum	cabeçalhos adicionados a cada requisição à fonte: playlist, segmentos, chaves
`variant_mode`	`single`	`single` — uma variante do master, `ladder` — a escada inteira
`variant_bandwidth`	nenhum	com qual bitrate casar a variante (o mais próximo abaixo); sem ele vence o maior `BANDWIDTH`
`skew_threshold_ms`	`500`	o desvio a partir do qual a escada é declarada fora de sincronia
`skew_policy`	`report`	o que fazer diante de um veredito `skewed`: `report` — continuar e mostrar, `fallback_to_single` — recuar para uma variante só
`connect_timeout_ms`	`5000`	tempo limite de estabelecimento da conexão
`read_timeout_ms`	`15000`	tempo limite do corpo: playlist, segmento, chave
`max_body_bytes`	64 MiB	limite de tamanho de uma única resposta
`peer_timeout_ms`	global	quanto esperar quadros antes de considerar a fonte perdida

Um campo omitido significa o padrão do servidor, não «desativado».

O esquema pode ser escrito de três formas, todas equivalentes:

- `https://host/path.m3u8` — reconhecido pela extensão do caminho, uma query não atrapalha;
- `hls://host/path.m3u8` — o mesmo que `http://`;
- `hlss://host/path.m3u8` — o mesmo que `https://`.

Regra de início

Tudo o que já estava na playlist no momento da conexão é história. A captura começa pelo primeiro segmento que aparece depois da conexão, e numa reconexão ela se comporta igual.

Não existe de propósito ajuste de profundidade de início: caso contrário, a cada queda de conexão o stream despejaria no pipeline, em segundos, a janela acumulada, em vez de transmitir.

Autenticação

Duas formas, e elas se combinam:

- **cabeçalhos** — `headers`, por exemplo `Authorization: Bearer <token>`; vão com a playlist, as variantes, os segmentos init, os segmentos e as chaves do mesmo origin, e nunca seguem um redirecionamento para um origin de terceiros;
- **credenciais na própria URL** — autenticação Basic comum feita pelo cliente HTTP.

Os valores dos cabeçalhos nunca chegam ao log: só os nomes são impressos. O endereço da entrada é escrito inteiro nos logs e devolvido como está pela API de gerenciamento — qual parte desse endereço é o segredo, só o operador sabe, e uma máscara às cegas ou deixaria o token na query, ou cortaria justamente aquilo pelo que a entrada é reconhecida no log.

Escada de qualidades e desvio das renditions

Com `variant_mode: ladder` captura-se toda variante do master. A de maior `BANDWIDTH` passa a ser a referência, e as demais são comparadas com ela segmento a segmento, pelo número de sequência de mídia. O veredito fica exposto na estatística da entrada, campo `hls_ladder`:

- **`synced`** — as renditions andam juntas, comutar entre elas é seguro;
- **`skewed`** — o desvio passou do limite ou a estrutura está quebrada: a comutação dará trancos ou dessincronizará;
- **`unmeasurable`** — não há com o que comparar, nenhum par de segmentos compartilha número.

O desvio é medido de duas maneiras ao mesmo tempo. O **declarado** é a diferença de `EXT-X-PROGRAM-DATE-TIME` na marcação; o **real** é a diferença do tempo de mídia dos primeiros quadros. Quando os dois discordam, o doente é a marcação, não o stream, e quem conserta é o dono da fonte.

> **Note**
>
> O Sapsan não traz as variantes da escada para uma escala de tempos comum, e não conserta de jeito nenhum um conteúdo ruim: um conserto silencioso não torna a escada comutável, apenas esconde o defeito. Por isso o desvio é mostrado como número, e quem decide é o operador.

A política `fallback_to_single` derruba a captura para uma variante só diante de um veredito `skewed` — para algumas fontes é o único modo viável. O recuo dura até a entrada ser reiniciada.

Criptografia

São suportados `AES-128` (o segmento inteiro) e `SAMPLE-AES` (por quadro, para H.264 e AAC). A chave é buscada pelo mesmo cliente e com os mesmos cabeçalhos que todo o resto, e fica em cache por endereço, de modo que a rotação de chaves no meio de uma playlist funciona sozinha — uma chave nova tem outro endereço.

Um servidor de chaves calado é um erro de entrada com o endereço da chave e o código de resposta; o corpo do segmento não é baixado de forma alguma, já que não haveria com o que descriptografá-lo.

CENC/DRM (Widevine, PlayReady, FairPlay) não é suportado e é rejeitado pelo nome do `KEYFORMAT`.

O que é suportado

Funcionalidade	Estado
Playlist de mídia: `EXT-X-MEDIA-SEQUENCE`, `EXTINF`, `EXT-X-ENDLIST`	sim
Playlist master: seleção de variante, renditions `EXT-X-MEDIA`	sim
Segmentos fMP4 (`EXT-X-MAP`) e MPEG-TS	sim
`EXT-X-BYTERANGE`	sim
`EXT-X-DISCONTINUITY`, recriação da playlist, ficar atrás da janela	sim
`EXT-X-GAP`	sim, pulado sem erro
`EXT-X-PROGRAM-DATE-TIME` — ancoragem do tempo de mídia ao UTC	sim
Criptografia `AES-128` e `SAMPLE-AES` (H.264, AAC)	sim
Escada: captura de toda variante, medição do desvio	sim
LL-HLS: `EXT-X-PART`, blocking reload	não, as partes são ignoradas
CENC/DRM	não
Áudio «empacotado»: ADTS puro ou ID3+AAC em vez de um contêiner	não, o formato é nomeado no erro
Legendas TTML e segmentos WebVTT	não, a faixa é reconhecida e pulada
Captura MPEG-DASH	não

Aplicação das mudanças

Só uma mudança de `url`, `headers` ou dos ajustes HTTP (`connect_timeout_ms`, `read_timeout_ms`, `max_body_bytes`) reinicia a entrada: uma fonte nova tem sua própria numeração de faixas, seu próprio init e sua própria posição na playlist. Todo o resto — modo de variante, limite e política de desvio, `peer_timeout_ms` — é aplicado em tempo real, sem interromper a transmissão.

Verificação

Abra `http://server/streaming/v/chan1/index.m3u8` num reprodutor ou peça uma captura de tela em `http://server/streaming/live-preview-jpeg/chan1`.

Vá além de «os quadros estão fluindo» e olhe três coisas:

- **o quanto a saída atrasa em relação à borda da playlist** — sem isso, a reclamação «o stream está atrasado» é indistinguível de «a fonte está atrasada», e as duas se consertam em lugares diferentes;
- **os segmentos pulados** e seu detalhamento: lavados para fora da janela, cancelados em voo, declarados com `EXT-X-GAP`. O último não é um erro da fonte, e sim a sua declaração honesta;
- **a emenda entre fragmentos vizinhos** — um defeito silencioso: todos os segmentos foram baixados, sem erros de análise, mas a mídia dentro deles não se encaixa, e o arquivo guarda o buraco.

Próximos passos

- Adicionar uma fonte reserva
- Gravar o stream no arquivo
- Transcodificar o stream
- Servir o stream por HLS

3.2.6 Canal a partir de um arquivo

A entrada `file` transforma um arquivo MP4 local num stream ao vivo de verdade: o arquivo toca em loop. É uma ferramenta de produção para canais de vinheta: um canal técnico de «voltamos já», um canal de promoção e um fallback para quando as fontes caem.

Configuração

```
streams:
  - name: promo
    inputs:
    - file:
        path: /storage/promo.mp4
    dvr: {}
```

O arquivo pode ser multi-faixa: várias qualidades de vídeo e várias faixas de áudio em um só MP4 dão um stream MBR completo com seleção de idioma — sem transcodificador no servidor.

Como preparar o arquivo com ffmpeg

Para um arquivo MBR tocar sem emendas, todas as qualidades devem estar perfeitamente alinhadas: timestamps idênticos e keyframes nos mesmos lugares. Isso é conseguido com uma única passada de ffmpeg usando o filtro `split`:

```
ffmpeg -y -i source.mkv \
  -filter_complex "\
[0:v:0]split=3[vhi][vme][vlo]; \
[vhi]scale=1920:1080:force_original_aspect_ratio=decrease,pad=1920:1080:(ow-iw)/2:(oh-ih)/2,setsar=1,fps=25,setpts=PTS-STARTPTS[v1]; \
[vme]scale=1280:720:force_original_aspect_ratio=decrease,pad=1280:720:(ow-iw)/2:(oh-ih)/2,setsar=1,fps=25,setpts=PTS-STARTPTS[v2]; \
[vlo]scale=960:540:force_original_aspect_ratio=decrease,pad=960:540:(ow-iw)/2:(oh-ih)/2,setsar=1,fps=25,setpts=PTS-STARTPTS[v3]" \
  -map "[v1]" -c:v:0 libx264 -b:v:0 6000k -g:v:0 100 -keyint_min:v:0 100 \
    -sc_threshold:v:0 0 -x264-params:v:0 "scenecut=0:open_gop=0" \
  -map "[v2]" -c:v:1 libx264 -b:v:1 3000k -g:v:1 100 -keyint_min:v:1 100 \
    -sc_threshold:v:1 0 -x264-params:v:1 "scenecut=0:open_gop=0" \
  -map "[v3]" -c:v:2 libx264 -b:v:2 1200k -g:v:2 100 -keyint_min:v:2 100 \
    -sc_threshold:v:2 0 -x264-params:v:2 "scenecut=0:open_gop=0" \
  -map 0:a:0 -c:a:0 aac -b:a:0 192k -ac 2 -ar 48000 \
  -metadata:s:v:0 title="1080p" -metadata:s:v:1 title="720p" -metadata:s:v:2 title="540p" \
  -metadata:s:a:0 language=eng \
  -movflags +faststart \
  promo.mp4
```

O que importa aqui:

- **uma única passada, uma única fonte,** `split` — todas as qualidades ganham timestamps idênticos;
- `setpts=PTS-STARTPTS` — a linha do tempo de cada qualidade começa no zero;
- o mesmo `fps` em todas as qualidades;
- **GOP rígido**: `-g` = `-keyint_min`, `-sc_threshold 0` e `scenecut=0:open_gop=0` — keyframes estritamente a cada N quadros e nos mesmos lugares; o codificador não pode inseri-los nas mudanças de cena;
- áudio AAC a 48 kHz estéreo;
- `-movflags +faststart` — o índice no começo do arquivo;
- `-metadata:s:v title=...` / `language=...` — as etiquetas de qualidade e os idiomas das faixas acabam nas playlists.

Um exemplo completo e funcional com várias faixas de áudio está no repositório do sapsan: `storage/king.sh`.

Verificação

Abra `http://server/streaming/v/promo/index.m3u8` — na playlist master devem estar todas as qualidades, e a comutação entre elas não deve engasgar.

Próximos passos

- Usar um arquivo como fallback quando uma fonte cai
- Fonte sintética de testes

3.2.7 Fonte sintética

A entrada `syntetic` é um gerador de sinal de teste embutido: barras SMPTE com um relógio correndo e um tom. Não precisa de dados externos e serve para verificar a instalação, depurar e fazer testes de carga.

Configuração

O gerador emite quadros sem compressão (YUV e PCM brutos). Isso é deliberado: o sinal bruto pode ser enviado sem perdas direto ao SDI. Para servir o stream por protocolos de rede (e obter as variantes de codec e qualidade de que precisa), combine o gerador com um transcodificador:

```
streams:
  - name: syn
    inputs:
    - syntetic: {}
    transcoder:
      output:
      - source: !content video
        codec: !set h264
      - source: !content audio
        codec: !set aac
    segments: 6
```

Sem uma seção `transcoder` o stream carrega quadros brutos: um reprodutor comum não consegue reproduzi-lo, mas é exatamente esse o stream necessário para a saída em SDI.

Parâmetros do gerador (todos opcionais, `syntetic: {}` dá um sinal de demonstração):

```
    - syntetic:
        video: !video
          rate:
            numerator: 30000     # frequências de quadro fracionárias (NTSC) são suportadas
            denominator: 1001
        audio: !audio
          sample_rate: 48000
          channels: 2
```

Timecode na imagem

O PTS de cada quadro é desenhado na banda de luma da imagem e pode ser relido por máquina. Isso permite medir a latência ponta a ponta e achar quadros perdidos pela própria imagem — útil na depuração do transcodificador e dos protocolos.

Próximos passos

- Transcodificação
- Canal a partir de um arquivo

3.2.8 Aceitação de publicações

Publicar significa que a fonte se conecta ao Sapsan por conta própria e entrega o stream (ao contrário da captura, quando é o Sapsan quem se conecta à fonte). O Sapsan aceita publicações por RTMP, SRT e WebRTC WHIP.

Configuração do stream

Um stream é declarado com uma entrada `publish`, que significa «esperar que alguém publique»:

```
listeners:
  rtmp:
    - port: 1935
  webrtc:
    - port: 5005

streams:
  - name: studio
    inputs:
    - publish: {}
```

Publicação por RTMP

Publique do OBS ou do ffmpeg para:

```
rtmp://server:1935/static/studio
```

> **Note**
>
> TODO: esclarecer o esquema do URL RTMP (application/stream key) e a autorização de publicação por streamkey.

Publicação por WebRTC (WHIP)

O endpoint WHIP padrão:

```
http://server/streaming/whip/studio
```

É possível publicar de um navegador ou de qualquer cliente compatível com WHIP (por exemplo, OBS 30+). Os codecs são H264 e Opus.

Um único publicador por stream

Enquanto uma publicação está ativa, uma segunda publicação no mesmo stream é rejeitada — o ar não pode ser interceptado. Se o stream também tem entradas comuns, uma publicação não expulsa uma fonte que funciona: o LSI comuta para o publicador segundo as regras habituais de prontidão (quadro-chave) e, com um arquivo de vinheta definido, a publicação pode ceder lugar à vinheta.

Publicação por SRT

A publicação por SRT é configurada como uma entrada SRT com uma porta dedicada por stream.

Autorização de publicações

As publicações são protegidas com `publish_tokens` — veja autorização.

Próximos passos

- Reproduza o stream publicado
- Grave a publicação no arquivo

3.2.9 Reserva de fontes

Um stream pode ter várias fontes. O Sapsan vigia a fonte ativa, comuta automaticamente para a próxima quando ela se degrada e volta para a principal quando ela se recupera. Esta lógica se chama LSI (Live Source Input).

Várias entradas

As entradas são listadas em ordem de prioridade — a primeira é a principal:

```
streams:
  - name: tv1
    inputs:
    - udp:
        host: 239.0.0.1
        port: 5000
    - tshttp:
        url: http://backup-origin.example.com/tv1/mpegts
```

Como funciona a comutação

Propriedades chave do algoritmo, importantes para o ar:

- **Uma fonte fica ativa apenas quando está pronta**: ela precisa entregar os parâmetros do stream (MediaInfo) e o primeiro quadro-chave. Até lá os espectadores continuam recebendo a fonte atual.
- **Retorno sem tela preta**: uma entrada de maior prioridade recuperada é sondada em segundo plano, e a comutação ocorre somente depois que ela entrega um quadro-chave.
- Uma entrada caída reconecta com pausas crescentes (`reconnect_initial_delay` → `reconnect_delay_step` → `reconnect_max_delay`); quando todas as entradas se esgotam, a rotação recomeça da primeira.
- Uma fonte cuja taxa de bits excede `max_bitrate` é interrompida à força.
- Tempos limite separados para a falta de vídeo e de áudio: o áudio perdido com vídeo vivo também conta como degradação.

Política de comutação

A seção `lsi` define a política para o stream inteiro; `source_options` numa entrada específica a sobrepõe:

```
streams:
  - name: tv1
    inputs:
    - udp: {host: 239.0.0.1, port: 5000}
    - udp:
        host: 239.0.0.2
        port: 5000
      source_options:
        source_timeout: 10
    lsi:
      source_timeout: 5
      video_timeout: 3
      audio_timeout: 3
```

Os tempos limite são definidos em segundos. Parâmetros principais (`lsi` e/ou `source_options`):

Parâmetro	Descrição
`source_timeout` , `video_timeout` , `audio_timeout`	tempos limite de ausência de dados/vídeo/áudio
`max_bitrate`	limite de taxa de bits da fonte
`reconnect_initial_delay` , `reconnect_delay_step` , `reconnect_max_delay`	política de reconexão
`retry_limit` , `max_retry_timeout`	limites de tentativas
`recheck_secondary_inputs_interval`	com que frequência verificar se uma entrada de maior prioridade voltou
`allow_if` , `deny_if` (apenas em `source_options`)	ligar/desligar uma entrada por um arquivo de bandeira externo

`allow_if` / `deny_if` permitem controlar as entradas de fora sem editar a configuração: uma entrada é usada apenas enquanto o arquivo de bandeira permite.

O arquivo de vinheta (backup)

Se todas as fontes caírem, em vez de uma tela preta é exibido um arquivo (veja como preparar o arquivo):

```
streams:
  - name: tv1
    inputs:
    - udp: {host: 239.0.0.1, port: 5000}
    lsi:
      backup:
        file: /storage/technical-difficulties.mp4
        timeout: 5
```

A vinheta entra em ação `timeout` segundos depois da queda das entradas (imediatamente, se não houver entradas nenhuma) e tem seus próprios `audio_timeout` / `video_timeout` .

Observabilidade

Por stream: o tempo na entrada principal e nas de reserva, o tempo sem dados, o número de tentativas, a validade das entradas reserva e um **detector de divergência de parâmetros** — se uma entrada reserva entrega parâmetros de vídeo diferentes (o que torna impossível uma comutação sem interrupção), isso fica visível com antecedência, antes do incidente. Por entrada: os tempos de abertura/conexão/início e o último erro. Veja monitoramento.

Próximos passos

- Prepare um arquivo de vinheta
- Monitore a comutação de fontes

3.2.10 Captura multi-bitrate

Muitas câmeras e codificadores entregam várias qualidades como streams separados. A entrada `mbr` os junta em um só stream multi-bitrate: o espectador recebe uma única playlist com todas as qualidades e comutação adaptativa.

Configuração

```
streams:
  - name: cam1
    inputs:
    - mbr:
        inputs:
        - rtsp:
            url: rtsp://admin:password@10.0.0.5/stream0   # qualidade alta
        - rtsp:
            url: rtsp://admin:password@10.0.0.5/stream1   # qualidade baixa
```

Dentro de `mbr.inputs` são permitidos os mesmos protocolos que nos `inputs` comuns.

> **Note**
>
> TODO: regras de alinhamento de faixas (timestamps, GOP), ordem das qualidades na playlist, comportamento quando uma das qualidades cai.

Alternativa: transcodificador

Se a fonte é única, o multi-bitrate é preparado pelo transcodificador.

Próximos passos

- Servir o stream MBR por HLS
- Gravar todas as qualidades no arquivo

3.2.11 Câmeras ONVIF

O Sapsan pode descobrir câmeras ONVIF na rede, receber seus eventos (movimento, detecção de pessoas e veículos) e verificar a compatibilidade de uma câmera.

> **Warning**
>
> O subsistema ONVIF está em desenvolvimento ativo — o conjunto de recursos e a configuração podem mudar.

Descoberta de câmeras

O Sapsan descobre as câmeras por WS-Discovery (ProbeMatch multicast) e adicionalmente entende o protocolo proprietário de descoberta da Hikvision. De cada câmera ele coleta os endereços (XAddrs), o nome e o modelo; observações repetidas da mesma câmera são mescladas.

Ao trabalhar através de NAT, os endereços que uma câmera informa sobre si mesma (URL de snapshot, URL de RTSP) são reescritos automaticamente para o endereço do dispositivo de fato alcançável.

Eventos da câmera

Os eventos dos metadados ONVIF viram episódios com abertura e fechamento:

- movimento (CellMotion `IsMotion`, MotionAlarm) — `true` abre um episódio, `false` o fecha;
- detecção de pessoas e veículos;
- disparo da entrada digital;
- episódios travados são fechados por um tempo limite de inatividade.

Várias fontes de movimento numa mesma câmera são rastreadas de forma independente.

Verificação de compatibilidade

A verificação integrada da câmera gera um relatório estruturado: informações do dispositivo, autenticação (HTTP Digest), relógio da câmera (o desvio de tempo é tratado automaticamente), classificação de eventos, além de um teste do stream RTSP com um resumo de erros (perda de pacotes, payload danificado, dessincronização) e estatísticas de quadros-chave.

> **Note**
>
> TODO: como disparar a descoberta e a verificação de compatibilidade (config/API/CLI), a configuração da câmera via ONVIF, a obtenção de snapshots.

Próximos passos

- Captura de uma câmera por RTSP

3.3 Transcodificação

3.3.1 Transcodificação

O transcodificador do Sapsan (baseado em FFmpeg) recodifica o stream de entrada: muda codecs, taxa de bits e resolução, e prepara uma escada de qualidades multibitrate (MBR) a partir de uma única fonte.

Configuração

A seção `transcoder.output` descreve as trilhas de saída. Cada trilha toma uma fonte (`!content video` ou `!content audio`) e define um codec:

```
streams:
  - name: tv1
    inputs:
    - udp: {host: 239.0.0.1, port: 5000}
    transcoder:
      output:
      - source: !content video          # 1080p — o topo da escada
        codec: !set h264
        bitrate: 2800000
        params: !video
          gop_size: 28
          gop_structure: ibbbp
      - source: !content video          # 720p
        codec: !set h264
        bitrate: 1200000
        params: !video
          gop_size: 28
          gop_structure: ibbbp
          resize: !fit
            height: 720
      - source: !content audio
        codec: !set aac
```

Seleção da trilha de origem

`source`	O que toma
`!content video` / `!content audio`	uma trilha por tipo de conteúdo
`!exact v1`	uma trilha específica por identificador
`!best_quality video`	a melhor qualidade disponível
`!language_codec [eng, aac]`	uma trilha por idioma e codec

O parâmetro `if_missing` define o comportamento quando a trilha não existe: `drop` (padrão — a saída não é criada), `blank` (sinal vazio) ou `substitute` .

Codecs e parâmetros

- `codec: !set <name>` — recodificar (h264, hevc, av1, aac, opus); `codec: same` — passar sem recodificar (por exemplo, apenas reempacotar o áudio).
- Saídas de codecs diferentes a partir de uma mesma fonte mantêm alinhados os timestamps e os quadros-chave — uma escada h264/hevc/av1 permanece consistente.
- `gop_size` , `gop_structure` (`ip` , `ibp` , `ibbp` , `ibbbp` , `ibbbbp`) — controle da estrutura do GOP, importante para segmentos MBR alinhados.
- Áudio: recodificação G.711 → AAC/Opus 48 kHz com preservação da linha do tempo — o caso típico de câmera IP.
- As trilhas JPEG de capturas atravessam o transcodificador intactas.

Redimensionamento

`resize`	O que faz
`!scale`	escala exata para as dimensões indicadas
`!fit`	encaixar preservando a proporção de aspecto (com preenchimento, cor de fundo configurável)
`!crop`	recortar para as dimensões indicadas

Note

TODO: aceleração por hardware (Nvenc/Vulkan), desentrelaçamento, overlays/logotipos, referência de parâmetros de áudio.

Verificação

Abra a playlist mestra `http://server/streaming/v/tv1/index.m3u8` — nela devem aparecer todas as qualidades de saída.

Próximos passos

- Distribua o stream transcodificado
- Grave a escada no arquivo

3.3.2 Legendas (speech to text)

O Sapsan pode reconhecer fala num stream gravado e escrevê-la numa faixa de legendas separada. O reconhecimento roda no motor Whisper embutido (whisper.cpp) **sobre o arquivo DVR**: um worker em segundo plano lê o áudio gravado, transcreve-o e acrescenta de volta no arquivo quadros de legendas. A faixa de legendas é então servida em HLS e DASH como WebVTT.

Como o worker lê do arquivo e não toca no stream ao vivo, ele não pode quebrá-lo: se o reconhecimento não dá conta, as legendas simplesmente atrasam e depois se põem em dia. Por isso o stream **precisa ter o DVR ligado**.

Pré-requisitos

1. **Uma compilação com a feature `whisper`.** O reconhecimento de fala enlaça uma pilha nativa whisper.cpp + ffmpeg, por isso vem desligado por padrão:

   ```
   bash
   make release FEATURES=whisper     # ou: cargo build --release --features whisper
   ```

   No Apple Silicon adicione Metal para aceleração por GPU (bem mais rápido): `--features whisper,metal`.
2. **DVR ligado** no stream (o worker lê e escreve o arquivo).
3. **Um arquivo de modelo.** Os modelos são nomeados por um nome curto (`tiny`, `base`, `small`, `medium`, `large-v3`); o Sapsan resolve o nome num arquivo GGML `ggml-<nome>.bin` no diretório de modelos:
   - `WHISPER_MODELS_DIR`, se definido, senão `~/.cache/whisper-models`.

   Baixe um modelo uma única vez, por exemplo `medium`:

   ```
   bash
   mkdir -p ~/.cache/whisper-models
   curl -L -o ~/.cache/whisper-models/ggml-medium.bin \
     https://huggingface.co/ggerganov/whisper.cpp/resolve/main/ggml-medium.bin
   ```

Configuração

Adicione um bloco `speech2text` a um stream que já tenha DVR:

```
streams:
  news:
    inputs:
    - rtsp:
        url: rtsp://admin:password@10.0.0.5/stream0
    dvr:
      root: /storage
    speech2text:
      model: medium        # tiny | base | small | medium (padrão) | large-v3
      language: ru         # ISO-639-1; omita para detecção automática
      audio_track: a1       # trilha de áudio de origem (padrão: a primeira trilha de áudio)
      max_window_secs: 30   # limite da janela de reconhecimento, s (padrão 30, máx. 120)
```

Campos:

Campo	Significado	Padrão
`model`	Modelo por nome curto; resolvido em `ggml-<nome>.bin`	`medium`
`language`	Idioma do reconhecimento (ISO-639-1)	auto
`audio_track`	Identificador da trilha de áudio de origem	a primeira trilha de áudio
`max_window_secs`	Limite superior da janela de análise	`30`

Validação: um stream com `speech2text`, mas sem `dvr`, é rejeitado; `max_window_secs` precisa estar em `1..=120`; `language` precisa ser um código de 2–3 letras.

As mudanças são aplicadas no reload (SIGHUP / central): o worker é reiniciado apenas se o bloco `speech2text` tiver mudado de fato.

Verificação offline (antes de ligar num stream)

O CLI `subtitle` (compilado com a feature `whisper`) roda exatamente o mesmo reconhecedor de produção num arquivo local, grava o resultado ao lado e, se o arquivo já tem uma faixa de legendas, calcula uma métrica de proximidade. É a forma mais rápida de escolher modelo e idioma e de avaliar a qualidade:

```
subtitle recognize movie.mkv --lang ru --model medium
# processar apenas um trecho (p. ex., 2 minutos a partir de 25:00) para uma verificação rápida:
subtitle recognize movie.mkv --lang ru --start-sec 1500 --limit-sec 120
```

Arquivos de saída, junto do arquivo de entrada:

- `movie.subtitles.json` — quadros estruturados de legendas com procedência completa (janela de áudio, modelo, hash, prompt, diagnóstico por cue), o bastante para reproduzir uma execução;
- `movie.vtt` — a projeção em WebVTT;
- `movie.diff.txt` — um lado a lado do texto de referência contra o reconhecido (quando o arquivo tem uma faixa de legendas de referência).

Se o arquivo tem legendas de referência, o CLI imprime **WER** (word error rate) e **CER** (character error rate), globalmente e por janela. Atenção: contra uma faixa de legendas feita de forma independente, o WER mede a divergência de tradução, não o erro de ASR — leia-o como um sinal relativo entre execuções e olhe o diff.

Qualquer arquivo `.mkv` / `.mp4` serve como entrada.

Verificação num stream

Depois de ligar o `speech2text` num stream com DVR:

1. As informações de mídia do stream ganham uma faixa de legendas. Verifique a playlist mestra do HLS — ela agora contém um rendition `EXT-X-MEDIA:TYPE=SUBTITLES`, e as variantes o referenciam com `SUBTITLES="subs"`:

   `bash`
   `curl -s "http://server/streaming/<stream>/index.m3u8" | grep -i subtitle`

   No DASH o manifesto ganha um `AdaptationSet` com `contentType="text"`.
2. Abra o arquivo num reprodutor com suporte a legendas (ou o URL de HLS/DASH) e ligue a faixa de legendas. As legendas aparecem com atraso, enquanto o worker alcança a borda ao vivo, e depois a acompanham.
3. Acompanhe os contadores do reconhecimento no endpoint de métricas (segundos processados, tamanho da lacuna / atraso, erros).

Como o worker é stateless — o progresso é derivado da borda da faixa de legendas no arquivo —, depois de um restart ele continua da mesma lacuna, completa com legendas o arquivo já acumulado e nunca duplica um quadro.

Como funciona

A unidade de trabalho do worker é a **lacuna** entre a cobertura do áudio e a cobertura das legendas no arquivo. Ele lê o áudio da lacuna, decodifica-o em PCM mono de 16 kHz, roda o Whisper em janelas deslizantes (com um gate de silêncio contra alucinações e reancoragem em segmentos completos) e acrescenta quadros estruturados de legendas. O WebVTT entregue aos reprodutores é uma projeção desses quadros; os quadros armazenados guardam adicionalmente procedência e diagnóstico do Whisper para depuração.

Próximos passos

- Configure a gravação DVR
- Reprodução HLS

3.3.3 Legendas ocultas

As legendas ocultas (closed captions, CC) viajam dentro do próprio vídeo, e não como uma faixa separada: nos dados auxiliares de cada quadro (SEI para H.264 e HEVC, user data para MPEG-2, metadados OBU para AV1). Elas são entendidas por TVs e decodificadores, mas um reprodutor de navegador não consegue mostrar esse texto: HLS e DASH esperam as legendas como uma faixa separada.

O Sapsan faz as duas coisas: passa as legendas ocultas dentro do vídeo como estão e as decodifica numa faixa de legendas WebVTT.

Modos de processamento

O modo é definido pelo campo `mode` da seção `closed_captions` do stream. Ele controla apenas a entrega — a decodificação e a gravação no arquivo acontecem sempre:

- **`passthrough`** (padrão) — as legendas ocultas ficam dentro do vídeo e são anunciadas nos manifestos como closed captions. Não há faixa de legendas nos manifestos.
- **`extract`** — uma faixa de legendas **no lugar** das legendas ocultas embutidas: o texto é decodificado numa faixa WebVTT, enquanto as legendas ocultas são cortadas do vídeo de saída e não são anunciadas como closed captions.
- **`both`** — as duas ao mesmo tempo: as legendas ocultas viajam dentro do vídeo e são anunciadas, e junto delas vai uma faixa de legendas.

Configuração

```
streams:
  - name: news
    inputs:
      - url: udp://239.0.0.1:1234
    closed_captions:
      mode: extract
      services:
        CC1:
          language: eng
          name: English
        SERVICE3:
          language: spa
          name: Espanol
```

A seção `services` é opcional e resolve duas tarefas:

- define o **nome e o idioma** do item de menu no reprodutor. Sem ela, o nome é deduzido do stream: o idioma anunciado pelo serviço e, caso contrário, o endereço do serviço (`CC1` , `SERVICE3`);
- **pré-anuncia** o serviço nos modos `extract` e `both` : um serviço listado ganha sua faixa desde o primeiro segmento, sem esperar o primeiro cue. Isso importa para reprodutores que leem a lista de faixas uma única vez na inicialização: um serviço que começa a falar no décimo minuto apareceria no menu só depois de reabrir o stream.

A chave do serviço é `CC1` … `CC4` para CEA-608 e `SERVICE1` … `SERVICE63` para CEA-708.

O que o reprodutor recebe

Cada serviço que fala vira sua própria faixa de texto com um número fixo: `CC1` … `CC4` → `t1` … `t4` , `SERVICE1` … `SERVICE63` → `t5` … `t67` . O número pertence ao serviço e não muda quando o stream reinicia, por isso os links para uma faixa continuam funcionando.

No HLS a faixa chega como um rendition `EXT-X-MEDIA` com `TYPE=SUBTITLES` ; no DASH, como um AdaptationSet com `contentType="text"` e `mimeType="text/vtt"` . Os segmentos da faixa ficam em endereços como:

```
/streaming/v/news/subtitles/t1/1738245600000.vtt
```

O último segmento do caminho é o início da janela em milissegundos; o fim da janela é decidido pelo servidor pela mesma grade de segmentos do vídeo. Uma janela vazia é um segmento WebVTT vazio válido, não um erro: numa pausa entre cues a faixa não pode se romper.

As legendas são servidas ao vivo, no rewind e do arquivo. Os cues decodificados são sempre gravados no arquivo, independentemente do modo — por isso ligar o `extract` funciona retroativamente: as legendas aparecem também no arquivo já gravado.

A limitação do modo `extract` nas saídas TS

No modo `extract` as legendas ocultas são cortadas do vídeo de saída por completo — em todas as saídas de uma vez, MPEG-TS incluso: push por udp/rtp/srt, segmentos `.ts` de HLS e tshttp. Numa faixa de texto não há como viajar dentro do MPEG-TS, então neste modo um consumidor de uma saída TS fica sem legendas de todo.

Se o mesmo stream é assistido num navegador e puxado por MPEG-TS ao mesmo tempo, use `both`: ele mantém as legendas ocultas dentro do vídeo para o consumidor do TS e adiciona a faixa para o navegador.

Migração do Flussonic Media Server

No Flussonic legacy a opção `cc.extract` **não** removia as legendas ocultas do vídeo — ela ligava a extração em adição às embutidas. O equivalente direto dela no Sapsan é, portanto, `mode: both`, e não `mode: extract`.

Flussonic	Sapsan	Comportamento
opção não definida	`mode: passthrough`	legendas ocultas dentro do vídeo, sem faixa
`cc.extract`	`mode: both`	legendas ocultas dentro do vídeo **e** uma faixa de legendas
—	`mode: extract`	comportamento estrito novo: uma faixa **no lugar** das legendas ocultas embutidas

`mode: extract` é um comportamento novo, sem análogo no Flussonic legacy. Escolha-o quando nenhum dos seus consumidores precisar das legendas ocultas embutidas: ele remove o item duplicado do menu do reprodutor (as mesmas legendas como closed captions e como faixa) e economiza espaço nos quadros de vídeo.

Verificação

Para confirmar que a faixa apareceu, verifique a playlist mestra:

```
curl -s http://localhost:8080/streaming/v/news/index.m3u8 | grep SUBTITLES
```

e o conteúdo do segmento:

```
curl -s http://localhost:8080/streaming/v/news/subtitles/t1/1738245600000.vtt
```

Os serviços detectados e o número da faixa de cada um aparecem nas estatísticas do stream (`GET /streaming/api/v4/streams/stats/news`, a seção `captions`).

O trabalho do pipeline é mostrado pelas métricas do Prometheus com o prefixo `stream_captions_`: `stream_captions_cc_pairs_total` — se as legendas ocultas chegam ao servidor de todo; `stream_captions_cues_decoded_total` — quantos cues foram decodificados; `stream_captions_cues_delivered_total` — quantos entraram na faixa. Um `stream_captions_unrecognized_payloads_total` crescente significa que o stream carrega um wrapper de legendas ocultas que não conhecemos; comunique isso ao suporte.

3.4 Reprodução

3.4.1 Reprodução por HLS e LL-HLS

Todo stream do Sapsan está disponível de imediato por HLS (fMP4/CMAF) sem configuração extra. Os streams ao vivo recebem por padrão LL-HLS com segmentos parciais.

URL de reprodução

O que	URL
Playlist mestre (todas as qualidades)	`http://server/streaming/v/<stream>/index.m3u8`
Playlist de uma única trilha	`http://server/streaming/v/<stream>/variant/<track>/index.m3u8`
Rebobinamento (rewind)	`http://server/streaming/v/<stream>/rewind/<segundos-atrás>/index.m3u8`
Arquivo por tempo absoluto	`http://server/streaming/v/<stream>/index.m3u8?from=<utc_ms>&to=<utc_ms>`

A playlist mestre traz as qualidades com `RESOLUTION`, `FRAME-RATE` e `CODECS`; o áudio vai para grupos rendition (`EXT-X-MEDIA`). Áudio G.711 (PCMA/PCMU) não é servido por HLS — transcodifique esses streams para AAC primeiro.

LL-HLS

Para os streams ao vivo o Sapsan serve um LL-HLS completo (versão 9 do protocolo):

- segmentos parciais `EXT-X-PART` na borda ao vivo e `EXT-X-PRELOAD-HINT` para a próxima parte;
- blocking playlist reload: os parâmetros de cliente `_HLS_msn` e `_HLS_part` seguram a requisição até aparecer o segmento ou a parte pedida;
- atualizações delta da playlist: `_HLS_skip=YES` encurta a playlist com a tag `EXT-X-SKIP`;
- `EXT-X-RENDITION-REPORT` para uma troca de qualidade rápida;
- `PART-HOLD-BACK`, `CAN-SKIP-UNTIL`, `HOLD-BACK` são calculados automaticamente a partir das durações do segmento e da parte.

O LL-HLS pode ser desativado por cliente com `?llhls=false` — a playlist mestre o propaga para todas as URLs das variantes, e o reprodutor recebe um HLS clássico (versão 7).

Tokens nas playlists

Se a reprodução estiver protegida por token, `?token=` é acrescentado automaticamente a todas as URLs aninhadas da playlist: variantes, segmentos init, segmentos, partes e preload hints. Ao reprodutor basta abrir a playlist mestre com o token.

Quantidade de segmentos

O comprimento da playlist ao vivo é definido pelo parâmetro `segments` do stream:

```
streams:
  - name: tv1
    inputs:
    - udp: {host: 239.0.0.1, port: 5000}
    segments: 6
```

Diagnóstico

O Sapsan traz embutido um validador de playlists HLS: ele estima a latência esperada e aponta problemas de configuração com códigos como `LL-DISABLED`, `TARGETDURATION-INFLATED`, `PART-TARGET-LARGE`, `SPARSE-INDEPENDENT-PARTS` (GOP longo demais para o LL-HLS).

> **Note**
>
> TODO: como executar o validador (CLI/API).

O que vem a seguir

- Proteja a reprodução com um token
- Reproduza o arquivo

3.4.2 Reprodução por DASH

Todo stream do Sapsan está disponível por MPEG-DASH: tanto ao vivo quanto o arquivo.

URL de reprodução

O que	URL
Manifesto ao vivo	`http://server/streaming/v/<stream>/Manifest.mpd`
Arquivo por tempo absoluto	`http://server/streaming/v/<stream>/Manifest.mpd?from=<utc_ms>&to=<utc_ms>`

Manifesto ao vivo

Um MPD dinâmico (perfil `isoff-live`): todas as qualidades de um stream MBR vivem em um único AdaptationSet de vídeo como Representation separadas, ordenadas por bitrate crescente — os reprodutores constroem corretamente a escada ABR. A janela `timeShiftBufferDepth` e o `suggestedPresentationDelay` são calculados automaticamente (o atraso é de uns dois segmentos atrás da borda ao vivo).

Manifesto do arquivo

O MPD do arquivo é estático. As lacunas da gravação tornam-se `<Period>` separados, e a linha do tempo é escolhida com o parâmetro `?timeline=`:

- `compact` (padrão) — as lacunas colapsam e o arquivo toca de forma contínua;
- `wallclock` — as lacunas são preservadas e a linha do tempo corresponde ao tempo real.

O `bandwidth` de cada Representation é medido a partir dos dados reais do intervalo solicitado. Se o arquivo tiver uma trilha JPEG de capturas, ela é exposta como um AdaptationSet padrão de thumbnail tile.

Diagnóstico

O validador de DASH embutido verifica um manifesto: o alinhamento dos tempos de início das trilhas da escada ABR, as lacunas e as sobreposições na SegmentTimeline entre atualizações, e o desvio da borda ao vivo entre áudio e vídeo. O modo (static/live) é detectado automaticamente a partir do `@type` do manifesto.

> **Note**
>
> TODO: como executar o validador, reprodutores verificados (dash.js, ExoPlayer).

O que vem a seguir

- Proteja a reprodução com um token
- Reproduza o arquivo

3.4.3 Saída MPEG-TS

O Sapsan multiplexa qualquer stream em um SPTS MPEG-TS e o serve por HTTP — prático para set-top boxes, transcodificadores e sistemas legados.

URL de reprodução

```
http://server/streaming/mpegts/<stream>
```

O que é produzido

- PAT/PMT/SDT corretas com contadores de continuidade contínuos; o número do programa, os PIDs e os nomes de serviço são definidos pela seção `mpegts` — veja Configuração da saída MPEG-TS; PIDs em conflito e reservados são reatribuídos automaticamente.
- Vídeo H264/HEVC, áudio AAC/AC3/EAC3, descritores de idioma das trilhas, teletexto (descritor teletext com páginas e idiomas).
- As trilhas JPEG de capturas nunca entram no TS.

CBR

O Sapsan pode entregar um bitrate estritamente constante — requisito dos receptores e moduladores profissionais:

- a taxa-alvo é derivada dos bitrates das trilhas (com folga de ~5% para o crescimento e um orçamento para o PSI);
- os pontos esparsos são preenchidos com pacotes nulos (PID 0x1FFF);
- o PCR é posicionado a partir de um relógio global sem deriva — a divergência entre PCR e DTS não se acumula nem com um conteúdo em loop de muitas horas;
- o modelo de buffer do decodificador (HRD) é respeitado — estouros e esvaziamentos são acompanhados.

> **Note**
>
> TODO: como ativar o CBR e definir o bitrate na configuração (hoje a taxa é derivada do bandwidth das trilhas).

Envio por UDP

Para enviar MPEG-TS para multicast por UDP, veja a página de retransmissão.

O que vem a seguir

- Capture MPEG-TS
- Retransmita o stream
- Configure a saída MPEG-TS

3.4.4 Configuração da saída MPEG-TS

Todas as saídas MPEG-TS de um stream — a reprodução por HTTP, SRT, os pushes por UDP, SRT e RIST — são produzidas por um único multiplexador. A seção `mpegts` do stream define as suas tabelas de serviço: o número do programa, os PIDs das trilhas e a descrição do serviço na SDT. A mesma seção num canal de saída individual sobrepõe esses valores apenas para esse canal: receptores diferentes recebem o mesmo stream sob números de programa e PIDs diferentes.

A seção mpegts do stream

```
streams:
  - name: tv1
    inputs:
    - udp: {host: 239.0.0.1, port: 5000}
    mpegts:
      pnr: 1030
      pmt_pid: 1000
      service_name: TV One
      provider_name: Example Media
      pids:
        v1: 1011
        a1: 1012
```

Todo parâmetro é opcional; um campo não definido significa o padrão do servidor:

Parâmetro	O que define	Padrão
`pnr`	número do programa: a entrada na PAT, `program_number` na PMT e `service_id` na SDT	`1`
`pmt_pid`	o PID da tabela PMT	atribuído automaticamente
`pids`	PIDs explícitos das trilhas por nome: `v1`, `a1`, …	atribuídos automaticamente
`service_name`	o nome do serviço na SDT	o nome do stream
`provider_name`	o nome do provedor na SDT	string vazia
`ts_stream_id`	`transport_stream_id` na PAT e no cabeçalho da SDT	`1`
`onid`	`original_network_id` no cabeçalho da SDT	`1`

Regras verificadas ao salvar a configuração:

- os **PIDs** devem estar na faixa 32–8190: valores abaixo são reservados às tabelas de serviço, acima, aos pacotes nulos;
- **PIDs duplicados** dentro da seção são rejeitados, inclusive um PID de trilha igual a `pmt_pid`;
- `pnr` nunca é zero — o número de programa zero na PAT é reservado à NIT;
- **os nomes do serviço e do provedor** juntos são limitados para que a SDT caiba em um único pacote TS.

Uma entrada em `pids` para uma trilha que o stream não tem agora é válida: ela vale quando a trilha aparece. Se um PID configurado explicitamente colide com o PID atribuído automaticamente a outra trilha, o valor explícito vence e a trilha desalojada migra de forma determinística para um PID livre — com um aviso no log.

SDT

Junto com a PAT e a PMT, a saída leva uma tabela SDT (PID 0x11): um único serviço do tipo digital television, `service_id` igual a `pnr`, e os nomes do serviço e do provedor tirados da seção `mpegts`. As strings são codificadas conforme o DVB: o latino vai como está e todo o resto como UTF-8 com a codificação declarada, por isso nomes não latinos são lidos corretamente pelos receptores.

Quando as configurações mudam, as versões das tabelas PAT/PMT/SDT afetadas são incrementadas e os receptores pegam as novas como devem. A edição da seção `mpegts` é aplicada em tempo real: pushes e clientes conectados não reconectam.

Configuração própria por canal

Os canais de saída individuais aceitam a mesma seção `mpegts`:

- um **push** por UDP, SRT ou RIST — o RTMP não tem MPEG-TS, por isso a seção não se combina com o transporte `rtmp`;
- `srt_play` — a reprodução SRT da porta dedicada do stream.

```
streams:
  - name: tv1
    inputs:
    - udp: {host: 239.0.0.1, port: 5000}
    mpegts:
      pnr: 1030
      service_name: TV One
    srt_play:
      port: 4010
      mpegts:
        pnr: 200
    pushes:
      local-multicast:
        udp:
          host: 239.1.1.1
          port: 5500
        mpegts:
          pnr: 300
          pids:
            v1: 211
            a1: 221
```

A seção do canal é fundida sobre a do stream campo a campo: um valor definido no canal sobrepõe o do stream, e um não definido é herdado. A exceção é o mapa `pids`: é um campo único, e um mapa não vazio do canal substitui inteiro o do stream — não há fusão por trilha.

A substituição está disponível apenas para os canais que o próprio stream envia. A reprodução MPEG-TS por HTTP e a porta SRT compartilhada do servidor servem sempre a saída comum do stream, com as configurações do nível do stream.

Como funciona

Para um canal com configurações próprias não se inicia um segundo multiplexador. O canal recebe uma cópia da saída comum na qual só as tabelas de serviço e os PIDs dos pacotes são reescritos: o arranjo dos pacotes no tempo, o PCR e os contadores de continuidade do multiplex comum são preservados, por isso as propriedades CBR da saída valem também para os canais sobrepostos. Canais com configurações efetivas idênticas dividem um stream byte a byte idêntico — cem pushes iguais custam tanto quanto um.

A versão de tabelas de uma saída reescrita é mantida pela sua própria história de mudanças, e nunca copiada da saída comum.

O que vem a seguir

- Reprodução MPEG-TS
- Reprodução SRT
- Push do stream

3.4.5 Saída SRT

Sapsan serve um stream por SRT: cada stream recebe uma porta UDP dedicada à qual se conectam os clientes SRT em modo caller.

Configuração

```
streams:
  - name: tv1
    inputs:
    - udp: {host: 239.0.0.1, port: 5000}
    srt_play:
      port: 4010
      passphrase: verysecretpass
```

`passphrase` ativa a criptografia; sem ela, o stream é servido sem criptografia.

A seção `srt_play` pode levar uma subseção `mpegts` — número de programa, PID das pistas e SDT próprios, só para a reprodução desta porta: configuração da saída MPEG-TS.

Reprodução

```
ffplay "srt://server:4010?passphrase=verysecretpass"
```

O que vem a seguir

- Capture um stream por SRT
- Retransmita o stream por SRT para outro servidor
- Configure a saída MPEG-TS

3.4.6 Saída RTSP

Sapsan funciona como servidor RTSP: qualquer stream pode ser puxado por RTSP — é a maneira padrão de alimentar de vídeo os gravadores NVR, os sistemas de analítica e outros servidores de mídia.

Configuração

Basta ativar o listener RTSP:

```
listeners:
  rtsp:
    - port: 554
```

Reprodução

```
ffplay rtsp://server:554/cam1
```

O que o servidor suporta:

- os métodos OPTIONS, DESCRIBE, SETUP, PLAY, TEARDOWN;
- o transporte RTP/AVP/TCP (interleaved) — funciona por uma única porta TCP, amigável aos firewalls;
- um RTCP Sender Report antes do primeiro pacote RTP de cada pista — o cliente obtém de imediato a ancoragem ao tempo absoluto;
- vídeo H264 e HEVC, áudio AAC (RFC 3640);
- as marcas de tempo originais do stream são preservadas.

> **Note**
>
> TODO: esquema exato do URL RTSP (caminho = nome do stream?), transporte UDP, autorização de reprodução por RTSP.

O que vem a seguir

- Capture um stream por RTSP

3.4.7 Reprodução por WebRTC (WHEP)

Para a reprodução com latência mínima, Sapsan serve os streams por WebRTC através do protocolo padrão WHEP.

Configuração

WebRTC precisa de uma porta UDP no servidor:

```
listeners:
  webrtc:
    - port: 5005
```

URL de reprodução

O endpoint WHEP do stream:

```
http://server/streaming/whep/<stream>
```

Serve qualquer reprodutor compatível com WHEP.

Requisitos e comportamento

- O stream deve ter uma pista de vídeo **H264**; o áudio **Opus** é opcional. Um stream com outro codec de vídeo não é servido por WHEP (recodifique com o transcodificador). Streams com áudio G.711 também passam pelo transcodificador para Opus.
- O servidor responde a PLI/NACK (um quadro-chave novo a pedido do reprodutor) e dosa a saída de pacotes em vez de inundar o cliente.
- WebRTC usa o conjunto fixo de portas UDP definido em `listeners.webrtc` — é fácil abri-las em um firewall; o endereço externo do servidor é detectado automaticamente.

> **Note**
>
> TODO: um exemplo de reprodutor HTML, limitações (ainda não há reprodução MBR, simulcast nem TWCC).

O que vem a seguir

- Aceite a publicação por WHIP

3.4.8 Capturas de tela

O Sapsan pode capturar periodicamente quadros JPEG de um stream: mostrar uma prévia atual nas interfaces e guardar os quadros no arquivo junto com o vídeo.

Configuração

Um stream pode ter vários geradores de capturas. Cada um toma os quadros de uma de duas fontes:

- `http` — baixar um JPEG pronto por HTTP (por exemplo, o snapshot-URL de uma câmera);
- `keyframe` — renderizar um quadro da própria trilha de vídeo do stream nos quadros-chave.

```
streams:
  - name: cam1
    inputs:
    - rtsp:
        url: rtsp://admin:password@10.0.0.5/stream0
    thumbnails:
    - http:
        url: http://10.0.0.5/snapshot.jpg
        fetch_interval: 10  # período de captura, s (padrão 10)
        timeout: 5          # tempo limite da requisição, s (padrão 5)
      dvr: true             # guardar os quadros no arquivo (padrão false)
    - keyframe:
        track: v1           # trilha de vídeo de origem
        width: 320          # padrão 320 se nenhum tamanho for definido; mínimo 32
        height: 180
        every: 3gop         # capturar a cada terceiro GOP (padrão: cada GOP)
```

Regras de validação: `fetch_interval`, `timeout` e `every` precisam ser maiores que 0; as dimensões do quadro devem ser de pelo menos 32; `timeout` é permitido apenas em `http`, e `width` / `height` / `every` apenas em `keyframe`.

Proteção contra lixo: uma resposta que não é um JPEG ou que excede o limite de tamanho é descartada — a prévia não é atualizada.

Obtenção de capturas

O que	URL
Quadro atual do stream	`http://server/streaming/live-preview-jpeg/<stream>`
Quadro de uma trilha específica	`http://server/streaming/track-preview-jpeg/<stream>/<track>`
Quadro do arquivo	`http://server/streaming/dvr-preview-jpeg/<stream>/<track>/image/<ref>`

Capturas no arquivo

Com `dvr: true` os quadros são gravados no arquivo como uma trilha JPEG separada com as dimensões reais do quadro. No manifesto DASH do arquivo ela é exposta como um AdaptationSet padrão de thumbnail tile (`http://dashif.org/guidelines/thumbnail_tile`) — reprodutores com suporte a miniaturas na linha do tempo o pegam automaticamente.

Próximos passos

- Configure a gravação DVR

3.4.9 Autorização

O Sapsan protege a reprodução e a publicação de streams. Há dois mecanismos: tokens estáticos na configuração e backends de auth externos a quem o servidor delega as decisões.

Como o token é passado

Um token é aceito de duas formas (o cabeçalho tem prioridade):

- o cabeçalho `Authorization: Bearer <token>`;
- o parâmetro de URL `?token=<token>`.

Para o HLS basta abrir a playlist mestra com o token — o próprio Sapsan acrescenta `?token=` em todos os URLs aninhados (variantes, segmentos, partes, init).

Tokens estáticos

```
auth:
  play_tokens:
  - viewer-secret-1
  publish_tokens:
  - publisher-secret-1
```

- `play_tokens` — tokens de reprodução;
- `publish_tokens` — tokens de publicação.

Uma requisição sem um token válido é negada. Se a seção `auth` existe, mas nem os tokens nem os backends corresponderam, o acesso é negado.

Backends de auth externos

As decisões podem ser delegadas a backends HTTP:

```
auth:
  upstreams:
    main:
      url: http://backend.example.com/auth
      timeout_ms: 3000
```

O Sapsan faz um `POST` para o `url` do backend com um corpo JSON:

```
{
  "kind": "play",                  // play ou publish
  "stream_name": "cam1",
  "proto": "hls",                  // protocolo: hls, dash, rtsp, m4f, ...
  "user_agent": "Mozilla/5.0 ...", // pode estar ausente
  "token": "viewer-secret-1",      // pode estar ausente
  "session_id": "..."              // identificador estável da sessão
}
```

A decisão é tomada pelo status HTTP da resposta (o corpo não é analisado):

- `2xx` — permitir;
- `401` ou `403` — negar (a negação é cacheada — uma requisição idêntica repetida não chega ao backend);
- qualquer outro status, a indisponibilidade ou o excesso de `timeout_ms` (padrão 1000 ms) marca o backend como caído;
- **vários backends são consultados em paralelo**: basta um «permitir»; só negações — negar; todos os backends caídos — erro de «backend caído» (não uma negação silenciosa).

Sessões

A sessão de um espectador é identificada por quatro elementos: stream, tipo (play/publish), IP, token — requisições repetidas do mesmo espectador não criam sessões novas. Conexões de longa duração (RTSP, WebRTC) são reautorizadas periodicamente: se o backend revogar o acesso, a sessão é fechada. Os contadores de sessões (abertas, negadas, autorizadas) estão disponíveis no monitoramento.

Acesso ao Admin API

O Admin API é protegido por uma seção separada `api_auth` — veja Admin API.

Próximos passos

- Configure a publicação

3.5 Retransmissão

3.5.1 Retransmissão (push)

Sapsan pode enviar um stream de maneira ativa: para as CDNs e plataformas de streaming por RTMP, para outro servidor por SRT ou RIST, ou para a rede por UDP (multicast/unicast). Um stream pode ter vários pushes ao mesmo tempo.

Configuração

Os pushes de um stream formam um mapa de «nome do push — configuração», não uma lista. O nome é único dentro do stream e serve de chave: é por ele que o push é encontrado na configuração, nas estatísticas e nas métricas. Um push tem exatamente um protocolo.

```
streams:
  - name: tv1
    inputs:
    - udp: {host: 239.0.0.1, port: 5000}
    pushes:
      youtube:
        rtmp:
          url: rtmp://a.rtmp.youtube.com/live2/xxxx-xxxx
      backup-dc:
        srt:
          host: 203.0.113.10
          port: 9000
          passphrase: verysecretpass
      local-multicast:
        udp:
          host: 239.1.1.1
          port: 5500
```

Protocolo	Parâmetros
`rtmp`	`url`, `connect_timeout_ms`
`srt`	`host`, `port`, `passphrase`, `stream_id`, tempos limite
`rist`	`host`, `port`, `cache_ms`, `retransmit_rate_percent`, `ttl`, `multicast_interface`
`udp`	`host`, `port` — MPEG-TS para multicast ou unicast

O nome é a identidade do push. Renomeá-lo não leva nem a configuração nem os contadores acumulados: o push antigo desaparece e o novo começa do zero.

Um push por UDP, SRT ou RIST pode levar a sua própria seção `mpegts` — número de programa, PID das pistas e SDT próprios, só para esse push: configuração da saída MPEG-TS.

Capacidades e comportamento

- O push RTMP suporta H264+AAC, HEVC e AV1 (enhanced RTMP), além de streams só de áudio; as marcas de tempo no push começam de zero, como as CDNs esperam.
- Os erros do push são distinguidos e ficam visíveis no status: conexão recusada, tempo limite de conexão, recusa de publicação (`NetStream.Publish.BadName` — chave ocupada ou errada), servidor travado (tempo limite de escrita de quadros).
- Um push SRT com uma `passphrase` divergente recebe uma recusa no handshake; um receptor silencioso (sem ACKs) é registrado como peer timeout.

RECONEXÃO

O pusher não tem política de novas tentativas própria — quem o levanta de novo é o stream. A cada cinco segundos o stream coteja os pushes em execução com a configuração e substitui um pusher morto por um novo; o contador `reconnects` conta exatamente essas substituições. Por isso o envio para um destino inalcançável nunca «desiste» nem precisa de reinício manual: ele continuará sendo levantado enquanto o push estiver ativado.

O que **não** conta como reconexão:

- **editar a configuração de um push** — o pusher também é reiniciado, mas é uma decisão do operador, não uma falha do canal;
- **desligar e ligar** (`enabled: false`) — desligar não é excluir: a configuração e os contadores acumulados permanecem, só o envio cessa.

DESATIVAÇÃO E EXCLUSÃO

Um push desativado permanece na configuração com o seu nome: as estatísticas o mostram como desativado, seus contadores ficam congelados nos últimos valores, e ao ligá-lo a contagem continua dali. Excluir a chave exclui também as estatísticas: os contadores pertencem ao nome.

> **Note**
>
> TODO: seleção de pistas para um push (qual qualidade sai).

Monitoramento

Cada push tem estatísticas próprias — destino, status, causa da falha, taxa, bytes, erros, reconexões. A referência de campos, as séries do Prometheus e as receitas de alerta estão na página Estatísticas.

O que vem a seguir

- Estatísticas de push
- Monitore os pushes
- Configure a saída MPEG-TS de um push

3.6 DVR

3.6.1 Gravação DVR

O DVR do Sapsan é um armazenamento próprio em disco: o stream é gravado append-only em arquivos por hora, por pista, em vários discos ao mesmo tempo. Ler o arquivo não atrapalha gravá-lo.

Armazenamento

A seção global `dvr` descreve o armazenamento inteiro:

```
dvr:
  root: /storage            # raiz do arquivo
  catalog: /storage/catalog # catálogo de blobs (por padrão <root>/catalog)
  disks:
  - path: d1                # discos relativos a root
  - path: d2
  - path: d3
```

Parâmetros do disco:

Parâmetro	Descrição
`path`	caminho do disco relativo a `root`
`mode`	`Active` (padrão) ou `Degraded` — o disco é lido, mas não se grava nele
`min_free_bytes`	reserva mínima de espaço livre

`check_mount` verifica que o caminho do disco seja de fato um ponto de montagem (protege contra gravar num diretório vazio quando um disco cai).

> **Warning**
>
> Os discos de gravação são listados explicitamente. A `root` é a casa do catálogo e a base das rotas relativas, não um disco: com `disks` vazio, a gravação de cada fragmento termina em erro, visível nas estatísticas, e nada é gravado na `root`.

Além dos discos de arquivo, o armazenamento pode ter discos de cache (`cache_disks`) — neles assentam as partes do arquivo que foram lidas: o alheio trazido de outro servidor ou o próprio levantado dos discos de arquivo. A gravação ao vivo nunca os escolhe, e o espaço neles não é liberado pela limpeza, e sim pelo despejo conforme a idade do último acesso. Isso é o cache do arquivo num nó edge.

Ativar a gravação num stream

```
streams:
  - name: cam1
    inputs:
    - rtsp:
        url: rtsp://admin:password@10.0.0.5/stream0
    dvr:
      max_depth: 168        # profundidade do arquivo, horas
      max_bytes: 500000000000
```

`dvr: {}` liga a gravação com as configurações padrão.

Distribuição da gravação entre discos

O Sapsan distribui sozinho a gravação entre os discos ativos:

- horas consecutivas de um mesmo stream alternam entre discos;
- streams vizinhos se espalham de forma parelha entre os discos;
- as pistas de um stream multi-bitrate são gravadas em paralelo em discos diferentes.

Um disco em modo `degraded` continua servindo leituras mas não recebe dados novos — é assim que um disco é aposentado sem perder o arquivo.

Limpeza e retenção

A limpeza roda uma vez por hora (5 minutos depois da virada da hora):

- `max_depth` age com granularidade de hora: só são apagadas as horas plenamente vencidas;
- `max_bytes` é contado dos blobs mais novos para os mais velhos: é apagado tudo mais antigo que o primeiro blob a exceder o limite;
- com os dois limites definidos, vence o mais rígido;
- blobs com episódios protegidos nunca são apagados pela retenção;
- o blob da hora atual (a que está sendo gravada) nunca é apagado.

A proteção contra estouro do disco funciona à parte: quando o espaço livre cai abaixo de `min_free_bytes` (por padrão, 1 % da capacidade do disco), o Sapsan apaga os blobs mais velhos **desconsiderando a retenção** até liberar espaço suficiente.

Como funciona o armazenamento

O arquivo está disposto nos discos como `<disk>/<hash>/<stream>/<Y>/<M>/<D>/<hour>.mp4`. Os ficheiros são apenas anexados, nunca reescritos; segmentos init idênticos são deduplicados dentro de um blob.

Um indexador em segundo plano reconcilia sem parar o catálogo com os discos e cura as divergências sem participação do administrador:

- blobs que apareceram no disco às costas do catálogo (por exemplo, depois de mover discos de outro servidor) são acrescentados ao catálogo;
- registros de ficheiros desaparecidos são removidos;
- o índice sidecar de um blob corrompido é reconstruído varrendo o próprio blob;
- os tamanhos em bytes e os filtros bloom dos blobs são preenchidos em segundo plano (blobs mais novos que uma hora não são mexidos).

As leituras toleram falhas: se o catálogo aponta para o disco errado, o fragmento é procurado em todos os discos daquela hora; registros órfãos do catálogo são pulados sem erro.

Observabilidade

O DVR entrega estatísticas por disco (espaço livre/total, contadores de operações e de bytes de leitura/gravação, número de blobs, bytes ocupados pelo arquivo) e do catálogo — veja o Admin API e o monitoramento.

Próximos passos

- Reproduzir o arquivo
- Fazer cache do arquivo alheio num edge
- Obter capturas de tela do arquivo

3.6.2 Reprodução do arquivo

O arquivo gravado está disponível pelos mesmos protocolos do ao vivo: HLS e DASH. Há ainda uma API para consultar as faixas gravadas e os previews.

A reprodução do arquivo funciona só em streams com gravação ligada (`dvr`) — caso contrário, a solicitação é rejeitada com um erro claro.

Retrocesso

Uma playlist retrocedida N segundos para trás a partir de agora:

```
http://server/streaming/v/<stream>/rewind/<seconds>/index.m3u8
```

Por exemplo, `rewind/3600/` é um timeshift de uma hora. O retrocesso une o arquivo ao ao vivo sem costura: a playlist começa no arquivo e continua no ao vivo, sem duplicatas nem buracos na junção; LL-HLS (blocking reload, atualizações delta) continua funcionando. Os buracos da gravação viram um `EXT-X-DISCONTINUITY` padrão.

Reprodução por tempo absoluto

O intervalo do arquivo é definido com os parâmetros `from` / `to` (UTC em milissegundos) nas URL comuns:

```
http://server/streaming/v/<stream>/index.m3u8?from=<utc_ms>&to=<utc_ms>
http://server/streaming/v/<stream>/Manifest.mpd?from=<utc_ms>&to=<utc_ms>
```

A playlist HLS do arquivo é uma playlist VOD fechada. No DASH, os buracos da gravação viram Periods, e a linha do tempo é escolhida com `?timeline=compact|wallclock` — veja DASH.

Quais faixas estão gravadas

Os intervalos gravados são entregues pela HTTP API:

```
http://server/streaming/dvr-range/<stream>
http://server/streaming/dvr-ranges/<stream>
```

> **Note**
>
> TODO: formato de resposta de dvr-range / dvr-ranges.

Previews de quadros do arquivo

- Um quadro JPEG da pista de capturas de tela: `http://server/streaming/dvr-preview-jpeg/<stream>/<track>/image/<utc_ms>.jpg`
- Um fragmento MP4 de um momento do arquivo: `http://server/streaming/dvr-preview-mp4/<stream>/<utc_ms>` — um fragmento curto a partir de um quadro-chave; para o preview, o servidor mesmo escolhe a pista de menor resolução.

Próximos passos

- Configurar a gravação
- Reproduzir um arquivo de outro servidor
- Proteger o arquivo com autorização

3.6.3 O arquivo de outro servidor (remotes)

O Sapsan sabe reproduzir sem emenda um arquivo que está gravado (ou já esteve gravado) em outro servidor. O stream lista os servidores remotos — e os arquivos deles se tornam uma extensão do local: o espectador vê um arquivo contínuo único e nunca sabe de onde os dados chegam fisicamente.

As tarefas típicas que isso resolve:

- migrar para um servidor novo sem perder histórico — o novo grava o arquivo fresco, o velho guarda o passado;
- ler o arquivo de servidores réplica;
- se recuperar depois da troca de um disco.

Configuração

```
streams:
  - name: cam1
    inputs:
    - rtsp:
        url: rtsp://10.0.0.5/stream0
    dvr: {}
    remotes:
    - url: http://old-server:5080
      timeout_ms: 5000
```

Parâmetro	Descrição
`url`	endereço do servidor Sapsan remoto onde está o arquivo deste stream
`timeout_ms`	tempo limite das requisições ao servidor remoto

Pode haver vários servidores remotos — o Sapsan consulta todos.

Como se consegue a emenda perfeita

A cada acesso ao arquivo o Sapsan une os dados locais e os remotos:

- **Bordas do arquivo** — a união das faixas de todos os servidores: o começo mais cedo e o fim mais tarde. Nas interfaces e na API o stream parece um arquivo contínuo único.
- **Playlists** — as listas de fragmentos do DVR local e de todos os servidores remotos são unidas, ordenadas e deduplicadas. Na junção de dois arquivos não há nem emenda, nem buraco, nem duplicata.
- **Fragmentos** — são lidos em cascata: buffer do ao vivo → DVR local → servidor remoto. A URL do fragmento é a mesma para o reprodutor independentemente de onde os dados moram.

Isso é possível graças ao endereçamento absoluto de fragmentos do Sapsan: o nome de um fragmento é a sua hora UTC, portanto o mesmo fragmento se chama igual em todos os servidores, e unir arquivos não exige recalcular a linha do tempo.

Replicação diferida

Um fragmento lido de um servidor remoto é gravado automaticamente no arquivo local. O DVR local vai «puxando» os pedaços de história alheia que os espectadores de fato assistem — e com o tempo deixa de buscá-los pela rede. Uma falha dessa gravação nunca atrapalha a entrega ao espectador.

A replicação grava a história alheia nos discos de **arquivo** e a mantém ali até a retenção limpá-la. Se o servidor não deve trazer o arquivo para si, e sim apenas economizar rede no que é popular, isso é o cache do arquivo: discos de cache separados, despejo pela idade do último acesso e funcionamento sem discos de arquivo algum.

Comportamento diante de falhas

- Se um servidor remoto está inacessível mas os dados existem localmente, a reprodução continua; o problema é apenas registrado no log.

- O Sapsan memoriza de quais faixas um remoto não dispõe (cache negativo) e não volta a consultá-lo por dados ausentes.
- Os fragmentos init dos streams remotos ficam em cache.

> **Note**
>
> TODO: autorização de requisições entre servidores, recomendações de tempos limite, limites (quantos remotes são razoáveis), interação com `config_external`.

Próximos passos

- Configurar a gravação DVR
- Fazer cache do arquivo alheio num edge
- Reproduzir o arquivo

3.6.4 Cache do arquivo num nó edge

Um nó que serve o arquivo alheio em vez do seu — por remotes ou por uma fonte legacy — vai à fonte por absolutamente cada fragmento. O cache do arquivo muda isso: o fragmento lido assenta num disco de cache local, e a solicitação seguinte da mesma parte do arquivo é servida localmente, sem vai-e-vem pela rede.

É assim que se monta um edge de CDN: o origin guarda o arquivo, os edges guardam o que os espectadores de fato assistem.

O cache é transparente. Ele muda de onde vêm os bytes, não o que é servido: as URL dos fragmentos, as playlists e as bordas do arquivo continuam exatamente as mesmas para o espectador.

O que distingue um disco de cache de um disco de arquivo

O disco de cache é um *papel* do disco, não uma flag nele:

- **a gravação ao vivo nunca o escolhe** — o arquivo é gravado apenas nos discos listados em `disks`;
- **a retenção do stream não o governa** — `max_depth` e `max_bytes` do stream não se aplicam ao cache;
- **o espaço é liberado pelo despejo** — pelo tempo desde o último acesso a um blob, não pela idade do conteúdo dele;
- **ele é isolado no catálogo** — os blobs de cache vivem em suas próprias tabelas, e as varreduras do arquivo (limpeza, backfill, a divisão do espaço ocupado em baldes) não os vêem em absoluto.

É por isso que o cache de um edge não aparece no console como um arquivo órfão, e por isso o despejo do cache não compete com a gravação do arquivo pelo mesmo mutex.

Configuração

O cache é configurado em dois lugares: os discos no nó, o interruptor no stream.

```
dvr:
  root: /storage
  cache_disks:
  - path: ssd1              # relativo a root, como um disco de arquivo
    max_bytes: 536870912000 # teto do cache neste disco, bytes
    min_free_bytes: 10737418240
  - path: ssd2

streams:
  - name: cam1
    remotes:
    - url: http://origin:5080
    cache: {}               # liga o cache das leituras do arquivo
```

Parâmetros do disco de cache:

Parâmetro	Descrição
`path`	caminho do disco relativo a `root`
`max_bytes`	quanto o cache pode ocupar neste disco; vazio — limitado só pelo espaço livre
`min_free_bytes`	quanto espaço livre deixar no volume

Os limites são independentes: o despejo roda até o disco caber nos dois. O disco de cache não tem `mode` — `Degraded` descreve um membro do RAID do arquivo, e o cache não replica nada.

A seção `cache` do stream é uma **seção de nível superior do documento, irmã da `dvr`**, não um campo dentro dela: cache e gravação são propriedades independentes. Um edge faz cache do arquivo alheio sem gravar o seu. A seção ainda não tem campos — a mera presença dela liga o cache.

A validação da configuração rejeita:

- um caminho listado ao mesmo tempo em `disks` e em `cache_disks`;
- uma duplicata dentro de `cache_disks`;
- um disco de cache cujo caminho seja igual a `root` — o despejo apagaria a casa do catálogo.

Caminhos diferentes no mesmo volume físico são uma configuração válida, mas na partida é registrado um aviso: o despejo libera espaço que o arquivo toma de volta na hora.

Uma seção `cache` num nó sem discos de cache é um no-op válido com um aviso no log. Isso é normal num cluster, onde um mesmo documento de stream se espalha por nós diferentes.

Um edge puro

Um armazenamento feito de `root` e discos de cache apenas, sem discos de arquivo, é uma configuração válida: há catálogo, não há gravação ao vivo, e o arquivo vai sendo trazido dos remotes.

```
dvr:
  root: /storage
  cache_disks:
  - path: ssd1
    max_bytes: 536870912000

streams:
  - name: cam1
    remotes:
    - url: http://origin:5080
    cache: {}
```

> **Warning**
>
> A `root` é a casa do catálogo, não um disco de gravação. Uma lista `disks` vazia costumava significar «gravar direto na `root`»; hoje é um erro de gravação visível nas estatísticas, em vez de uma gravação silenciosa por fora do RAID. Se o nó deve gravar um arquivo, liste os discos explicitamente em `disks`.

Admissão no cache

Pôr no cache tudo na primeira solicitação significa expulsar o popular em favor do aleatório: um único salto para o fundo do arquivo empurraria para fora do disco o que todos estão assistindo. Por isso a admissão tem dois eixos, e são configurados no nó, para o cache inteiro:

Parâmetro	Descrição
`cache_fresh_age_secs`	um fragmento mais novo que essa profundidade (segundos para trás a partir de agora) vai para o cache já na primeira solicitação; por padrão, um dia
`cache_admission_hits`	com qual solicitação repetida começa a ser guardado em cache um fragmento mais velho que o limite de frescor; por padrão, a segunda; `1` desliga a barreira

```
dvr:
  root: /storage
  cache_fresh_age_secs: 86400
  cache_admission_hits: 2
  cache_disks:
  - path: ssd1
```

Quase todo mundo assiste à cauda fresca do arquivo, então ela entra no cache na hora. O fundo do arquivo interessa a uma minoria — entra no cache só quando é pedido de novo.

Ordem das fontes de leitura

Um fragmento do arquivo é procurado ao longo de uma cadeia, e a primeira fonte que responde fecha a solicitação:

1. o segmentador do stream ao vivo;
2. **o cache**;
3. o arquivo local;
4. o cluster (remotes);
5. a fonte legacy `old_m4f`;
6. o arquivo legacy em disco.

Um acerto de cache responde sem tocar em nada abaixo dele. Se um fragmento está num disco de arquivo e num disco de cache ao mesmo tempo, a leitura é servida pelo cache: cache em SSD ganha dos discos mecânicos. Um stream sem seção `cache` não tem nenhum passo de cache na cadeia.

O que uma falha põe no cache

Uma leitura bem-sucedida de uma fonte abaixo do cache é gravada num disco de cache junto com o seu fragmento init. As diferenças em relação à gravação do arquivo:

- **o corte de `max_depth` não se aplica** — no cache entra o que o espectador pediu, mesmo que seja mais fundo que qualquer profundidade de arquivo configurada;
- **um segmento multipista é guardado inteiro, com todas as pistas** — assim a troca de bitrate de um espectador é servida por acertos, e não por falhas;
- **a gravação é idempotente** — vários espectadores falhando ao mesmo tempo no mesmo fragmento deixam exatamente uma cópia no cache;
- **um erro de gravação no cache não afeta a resposta do cliente** — ele já tem os bytes; o erro é apenas registrado e contado nas métricas.

O texto cifrado é guardado no cache como está, junto com o seu fragmento init: o edge não tem com o que descriptografá-lo e entrega exatamente os bytes que chegaram. Essa mídia é reconhecida pela descrição da sua trilha — o esquema e o identificador da chave chegam ao `MediaInfo` pelo fragmento init e fazem parte da identidade da trilha assim como o codec, de modo que uma troca de chave é resolvida pelo mesmo mecanismo que uma troca de codec. O texto cifrado nunca chega ao arquivo: o arquivo alimenta prévias, miniaturas e a linha do tempo, e todas elas leem dentro da amostra.

Leitura antecipada

Assistir ao arquivo é sequencial, por isso depois de servir uma leitura o Sapsan busca em segundo plano o próximo fragmento da mesma pista, se ele ainda não está no cache. A leitura antecipada funciona num edge puro e na leitura de uma fonte remota, limita-se a um fragmento à frente e nunca atrasa a resposta ao cliente.

Isso não é aquecimento: aqui não existe encher o cache por agenda nem por EPG.

Despejo

O espaço num disco de cache é liberado em blobs horários inteiros, na ordem do último acesso — o menos recentemente usado vai primeiro — até o disco voltar aos dois limites.

- A idade do conteúdo não é critério: o programa popular de ontem sobrevive ao impopular de hoje, e a hora atual é expulsa em igualdade de condições com as demais.
- A única exceção é um blob no qual um escritor está anexando dados agora mesmo: ele fica, e no lugar dele vai o seguinte menos recentemente usado.
- A limpeza do arquivo (`max_depth` e `max_bytes` do stream) e a proteção de episódios não consideram os discos de cache em absoluto.

A ordem de despejo é mantida por um índice próprio do catálogo e sobrevive a um reinício. A marca de acesso vai para o catálogo de forma diferida — no máximo cerca de uma vez a cada 10 minutos por blob — para que a leitura não vire gravação.

A contabilidade do cache (volume ocupado, tempos de acesso) é restaurada na partida por uma varredura de intervalos do catálogo, sem percorrer os ficheiros de dados. Um disco de cache apagado é um estado frio válido: o servidor parte, o cache se enche do zero, o arquivo fica intacto.

Linha do tempo e profundidade num edge

A profundidade do retrocesso e as faixas do arquivo de um stream com o cache ligado são calculadas como a união do catálogo local (arquivo e cache) e de todas as fontes remotas — e são anunciadas nas playlists de retrocesso e nas respostas sobre as faixas do arquivo. Um espectador no edge vê a mesma profundidade que no origin.

As janelas das fontes são unidas **em lista, preservando os buracos entre elas**: um intervalo sólido do começo ao fim de uma fonte não pode substituir essa lista — caso contrário a linha do tempo pinta de verde trechos sem dados, e um clique neles cai num 404. Buracos não maiores que a resolução pedida continuam sendo unidos pela regra geral.

Se o origin está inacessível mas a faixa pedida está inteira no cache, a playlist é construída a partir do catálogo local e os fragmentos são servidos do cache — a reprodução continua.

Observabilidade

Cada **leitura do arquivo** é atribuída ao elo da cadeia que a serviu:

- **cache** — serviu o disco de cache do nó (um acerto);
- **local** — serviu o disco de arquivo do nó;
- **remote** — foi preciso ir a uma fonte remota (uma falha).

Somente a leitura de um fragmento pelo cliente conta como leitura do arquivo. A saída do segmentador ao vivo e as buscas de leitura antecipada não entram na atribuição — caso contrário, a leitura antecipada inflaria a taxa de acertos quanto melhor funcionasse; as buscas são contadas à parte.

Para o Prometheus e o local-prom vão os contadores cumulativos:

Métrica	Significado
`dvr_archive_reads_total{source}`	leituras do arquivo pelos eixos `cache` / `local` / `remote`
`dvr_archive_read_bytes_total{source}`	bytes dessas leituras
`dvr_read_ahead_fetches_total`	buscas de leitura antecipada
`dvr_disk_reads_total{disk_id}`, `dvr_disk_read_bytes_total{disk_id}`	acessos ao disco; num disco de cache são os seus acertos
`dvr_disk_cache_bytes{disk_id}`	volume de cache no disco

O endpoint local `GET /streamer/api-v4/dvr` informa o papel, o volume e o limite de cada disco de cache, e as taxas de leitura do nó nos três eixos. Os contadores cumulativos não são expostos na management API: lá vão taxas e gauges.

A taxa de acertos nunca chega como número pronto — é `cache / (cache + local + remote)`, e quem calcula é o consumidor: as taxas se somam, a divisão vem depois. Uma média de médias mente quando os nós têm pesos diferentes.

As telas do console onde isso aparece estão descritas na documentação do Catena: capacidade de DVR no cluster e configurações do nó; a montagem de um edge em um cluster do Catena — Streamers edge.

Em que isso se diferencia da replicação diferida por remotes

Os remotes têm um comportamento parecido: um fragmento lido de um servidor remoto é anexado ao arquivo local. A diferença é fundamental:

	Replicação diferida por remotes	Cache do arquivo
Onde é gravado	em discos de arquivo	em discos de cache
Para quê	trazer para cá para sempre a história alheia	servir localmente as solicitações repetidas
Quando é liberado	pela limpeza segundo a retenção do stream	pelo despejo segundo o último acesso
Discos de arquivo necessários	sim	não, um edge pode funcionar sem eles

A replicação consiste em mudar o arquivo de lugar; o cache, em economizar rede no que é popular.

O que vem a seguir

- O arquivo de outro servidor (remotes)
- Gravação DVR
- Reprodução do arquivo

3.7 Administração

3.7.1 Admin API

O Sapsan expõe o estado do servidor por HTTP. A API é compatível em formato com a Flussonic Streamer API v3, então as integrações existentes continuam funcionando.

Acesso

O acesso à API é protegido por um login e uma senha na seção `api_auth`:

```
api_auth:
  login: admin
  password: secret
```

Endpoints

O prefixo base é `/streamer/api/v3`:

Método e caminho	O que retorna
`GET /streamer/api/v3/streams`	lista de streams e seu estado
`GET /streamer/api/v3/config`	configuração atual do servidor
`GET /streamer/api/v3/dvrs`	lista de armazenamentos DVR
`GET /streamer/api/v3/dvrs/{name}`	estado de um DVR específico
`GET /streamer/api/v3/rproxy`	estado do gateway Peeklio e dos agentes
`GET /streamer/api/v3/monitoring/readiness`	sonda de readiness para orquestradores

API nativa v4

A API nativa, ainda em evolução, vive sob o prefixo `/streamer/api-v4`:

Caminho	O que retorna
`GET /listeners`	lista de listeners
`GET /streams`, `GET /streams/{name}`	streams e seu estado
`GET /streams/stats`, `GET /streams-stats`	estatísticas de streams
`GET /sessions/stats`	estatísticas de sessões
`GET /live-metrics`, `GET /sessions-metrics`	métricas de live e de sessões
`GET /dvr`, `GET /dvr/catalog`, `GET /dvr/metrics`	estado do armazenamento e do catálogo DVR
`POST /dvr/rebuild-index`, `GET` (status), `DELETE` (parar)	reconstrução do catálogo a partir dos discos
`POST /dvr/cleanup`, `GET /dvr/cleanup`	iniciar a limpeza do arquivo / status
`GET /runtime/metrics`	métricas do processo em formato Prometheus
`GET /auth`	estado da autorização
`GET /license/status`	estado da licença
`GET /rproxy/streampoint-agents`	agentes do gateway Peeklio

> **Note**
>
> TODO: exemplos de respostas, gestão de streams pela API (quando surgirem operações de escrita).

O que vem a seguir

- Gestão externa da configuração
- Monitoramento

3.7.2 Gestão externa da configuração

Em instalações em cluster os streams do Sapsan são gerenciados por um sistema externo (por exemplo, o Flussonic Central): o servidor periodicamente busca numa URL externa a lista de streams e a aplica.

Configuração

```
config_external:
  url: http://central.example.com/central/api/v3/streamers/streamer1.example.com
  bearer: <token de acesso>
  client_host: streamer1.example.com
  interval_secs: 5
  timeout_ms: 30000
```

Parâmetro	Descrição
`url`	de onde buscar a configuração
`bearer`	token de autorização para o servidor externo
`client_host`	o nome com o qual este servidor se apresenta ao sistema de gestão
`interval_secs`	período de sondagem (5 s por padrão)
`timeout_ms`	tempo limite da requisição (30 s por padrão)
`fetch_runtime_config`	se buscar também a configuração de runtime (`true` por padrão)

Important

Quando o `config_external` está ligado, a seção local `streams` do `sapsan.yaml` é ignorada — a fonte da verdade passa a ser o servidor externo. As demais seções (`listeners` , `dvr` , `auth` , `api_auth`) continuam locais, a menos que o servidor externo entregue as suas.

Como funciona a sondagem

- O Sapsan pede `GET <url>/streams` a cada `interval_secs` ; a lista é trazida página por página pelo cursor `next` . Cada requisição leva `Authorization: Bearer` , um cabeçalho `x-originator: sapsan` e o `client_host` .
- A configuração de runtime (`GET <url>/config` — listeners, DVR, auth) é pedida só quando o servidor anuncia que tem uma.
- Se a configuração não mudou, a reaplicação é omitida.

Tolerância a falhas

- **Uma lista parcialmente inválida**: os streams quebrados são descartados (com indicação de qual campo falhou) e os válidos são aplicados — status `Partial` .
- **Uma configuração totalmente inválida** (por exemplo, conflito de portas): nada é aplicado, continua rodando a configuração anterior — status `Error` .
- Indisponibilidade do servidor externo, JSON quebrado e erros HTTP são distinguidos no status (`network` / `malformed_json` / `http<code>`).

O que vem a seguir

- Admin API

3.7.3 Gateway Peeklio (rproxy)

O Peeklio é um gateway de proxy reverso embutido no Sapsan. Um agente leve, instalado ao lado da câmera (atrás do NAT), mesmo se conecta ao Sapsan e mantém um túnel; por esse túnel o Sapsan busca o vídeo e o tráfego do dispositivo como se ele estivesse na rede local.

Configuração do gateway

```
rproxy:
  streampoint_key: <chave para conexões streampoint>
  endpoint_auth_url: http://backend.example.com/agent-auth
  agents:
  - agent_id: a1b2c3
    password: <senha do agente>
    salt: <sal>
    streampoint_url: http://this-server:5080
```

Os agentes podem ser autorizados de duas formas: com uma lista local `agents` na configuração ou com um backend externo `endpoint_auth_url`. Sem `endpoint_auth_url` o gateway funciona no modo «só streampoint» — aceita túneles, mas não registra agentes novos.

No lado do agente são definidos o endereço do servidor, o `agent_id`, a senha, o intervalo de pings e uma **ACL de hosts** — a lista de endereços que podem ser alcançados por meio deste agente; todo o resto é recusado.

> **Warning**
>
> O estado do gateway sobrevive às recargas de configuração (os agentes não caem no SIGHUP), mas trocar `streampoint_key` / `endpoint_auth_url` exige reiniciar o processo.

Captura de um stream por um agente

Na entrada de um stream indica-se `via_agent` com o identificador do agente:

```
streams:
  - name: cam-behind-nat
    inputs:
    - rtsp:
        url: rtsp://admin:password@192.168.1.10/stream0
        via_agent: "agent://a1b2c3"
```

O endereço em `url` é o endereço local da câmera na rede do agente. O handshake RTSP e o stream passam pelo túnel.

Comportamento diante de erros

Os erros de conexão por meio de um agente são distinguíveis: porta fechada (connection refused), host fora da ACL do agente (permission denied), nome que não resolve. O servidor pode enviar remotamente a um agente os comandos `reset` e `reboot`.

Monitoramento de agentes

`GET /streamer/api-v4/rproxy/streampoint-agents` traz contadores por agente: pings, bytes nos dois sentidos, tentativas/aberturas/conexões atuais. Veja o Admin API.

> **Note**
>
> TODO: instalação e configuração do próprio agente (pacote rproxy-agent, agent.toml), emissão de chaves.

3.7.4 Estatísticas

O Sapsan sempre coleta o conjunto completo de contadores — a licença e a tarefa só determinam por qual canal você os lê. São quatro canais:

Canal	O que lhe dá	Disponibilidade
Admin API	um retrato instantâneo em JSON: o que está acontecendo com o stream agora mesmo	todos
Servidor Prometheus embutido	um datasource Grafana pronto, com algumas horas de histórico — gráficos sem implantar uma TSDB	todos
Scrape do Prometheus	métricas cruas para a sua própria TSDB e Grafana	a opção de contadores estendidos
Retroview	o monitoramento em nuvem do fabricante: meses de histórico, tendências, alertas prontos, conciliação de consumo	assinatura

A regra de escolha é simples:

- «O que está acontecendo **agora**?» — a Admin API ou a UI.
- «O que aconteceu nas **últimas horas**?» — o servidor Prometheus embutido.
- «O que aconteceu **na semana passada** e por que caiu de madrugada?» — Retroview.
- «Quero **meu próprio** Prometheus, Grafana e alertmanager» — a opção de contadores estendidos.

Estatísticas do stream na Admin API

A LISTA DE STREAMS

`GET /streamer/api-v4/streams/stats` traz um elemento por stream — a página de streams da UI vive dessa lista, e ela também é o certo para sondá-la de scripts.

Campo	O que significa	Como usar
`status`, `status_since`	o estado do stream e o momento em que entrou nele	fora de `running` por mais de N minutos — alerta
`bitrate_kbit`	o bitrate de mídia medido da entrada ativa, pelos timestamps dos quadros	despencou bem abaixo do nominal — a fonte se degradou. Isso não é o throughput de rede: um arquivo pode ser lido mais rápido que o tempo real
`source.url`	a entrada que está tocando **de fato**	difere da primeira do config — o stream vive numa reserva
`inputs[]`	o status de runtime de cada entrada configurada: `url`, `status`, idade da verificação `checked_ago_ms`, bitrate, texto do erro	veja abaixo
`dvr.from`, `dvr.to`	a janela do arquivo neste servidor	`to` fica atrás da hora atual — a gravação parou; `from` parou de avançar — a limpeza não funciona, transbordamento de disco à frente
`bytes_in`, `bytes_out`	contadores cumulativos de bytes desde o início do stream	para taxas e histórico use o servidor Prometheus embutido ou o Retroview: o retrato zera no restart

Status dos elementos de `inputs[]`:

- `active` — tocando agora;
- `ok` — uma reserva, viva na última verificação;
- `error` — uma reserva com um problema registrado (texto em `error`);
- `unchecked` — uma reserva que ainda nunca foi verificada.

Isso responde à principal pergunta do failover — «vamos comutar de verdade quando a principal morrer?». As reservas são verificadas periodicamente (`recheck_secondary_inputs_interval`, veja fontes reserva); alerte sobre `error`, sobre `unchecked` e sobre um `checked_ago_ms` velho demais — uma reserva não verificada não pode ser considerada operacional.

UM ÚNICO STREAM

`GET /streamer/api-v4/streams/stats/{name}` acrescenta a esses mesmos campos seções de diagnóstico — elas ficam de fora da lista para que a lista continue barata.

A seção `input` — contadores da entrada atual:

Campo	Para quê
`bytes`, `frames`, `bitrate_kbit`	se a entrada traz dados e a que velocidade
`retries`	reconexões à fonte: cresce — a fonte ou a rede está instável
`input_switches`	comutações entre entradas: cresce — a entrada cai e volta; investigue a principal
`errors`	o contador agregado de erros de entrada: cresce — há um problema; a causa está nos contadores estendidos ou no Retroview
`num_sec_on_primary_input`, `num_sec_on_secondary_input`	segundos vividos na principal e na reserva: uma fatia notável na reserva — a entrada principal é sistematicamente ruim
`num_sec_no_data`	segundos sem dados nenhum
`errors_lost_packets`, `errors_connection_closed`, `errors_http_request_error`, …	detalhamento causal dos erros — **somente com a opção de contadores estendidos**

A seção `dvr` na resposta individual é ampliada com:

Campo	Para quê
`recorded_hours`	horas que realmente contêm dados. Compare com a largura da janela `to - from`: a diferença são buracos na gravação (o stream caiu, a gravação estava desligada)
`bytes`	tamanho total do arquivo do stream em disco
`write.segments_written`, `write.segments_failed`, `write.segments_skipped`	contadores de escrita de fragmentos: `segments_failed` cresce — o arquivo está perdendo dados agora mesmo, verifique o disco
`write.segments_written_slow / _delayed / _collapsed`, `write.segments_discontinuity`	perfilamento da escrita — **somente com a opção de contadores estendidos**

> **Note**
>
> O retrato da Admin API zera quando o servidor reinicia. Ele responde «o que está acontecendo agora», mas não serve nem para conciliar a conta nem para analisar o incidente de ontem — para isso existe o Retroview.

Estatísticas de pushes

A entrega de saída é contada à parte da captura e por push: a chave da estatística é o par «nome do stream, nome do push», o mesmo da configuração. Um stream pode sair para cinco destinatários, e «o stream envia 40 Mbit/s» não responde se ele chega a algum deles em particular.

> **A opção «estatísticas de pushes»**
>
> A vista de pushes é uma opção de licença à parte. Ela corta apenas os canais de saída: a seção `pushes` e os campos planos `push_*` na Admin API, as séries `stream_push_*` no scrape do nó e no servidor Prometheus embutido. A coleta em si nunca para, e os contadores sempre vão para a telemetria do Retroview — a análise no Retroview está disponível também sem a opção.

A SEÇÃO `pushes`

`GET /streamer/api-v4/streams/stats/{name}` (e cada item de `GET /streamer/api-v4/streams/stats`) traz um objeto indexado pelo nome do push:

```
"pushes": {
  "cdn-main": {
    "status": "running",
    "proto": "udp",
    "url": "udp://127.0.0.1:5500",
    "opened_at": "2026-08-13T11:17:53Z",
    "bitrate_kbit": 2113.4,
    "bytes_total": 19053048,
    "datagrams_total": 14478
  },
  "youtube": {
    "status": "error",
    "proto": "rtmp",
    "url": "rtmp://198.51.100.7/live/secret-key",
    "error": "rtmp connect failed: timeout after 5s",
    "reconnects_total": 22
  },
```

```
  "backup-dc": {"status": "disabled", "proto": "srt", "url": "srt://203.0.113.10:9000"}
}
```

Campo	O que significa	Como usar
`status`	`running`, `error` ou `disabled`	fora de `running` por mais de N minutos — alerta; `disabled` é normal, é um push que o operador desligou
`error`	a causa do status `error`	o primeiro a ler num diagnóstico: «connect failed», recusa de publicação, handshake rejeitado
`proto`, `url`	o transporte e o destino exatamente como o operador os definiu	dá para ver para onde o stream vai de fato
`opened_at`	o momento em que o pusher atual começou	fica mais novo a cada reconexão — o canal está caindo e voltando
`bitrate_kbit`	a velocidade de envio medida	bem abaixo do bitrate do stream — o envio não dá conta
`errors_rate`	erros de envio por segundo	cresce com a conexão viva — o receptor ou o canal estão degradando
`bytes_total`	bytes enviados, acumulado	conciliação do volume de saída por direção
`errors_total`	erros de envio, acumulado	—
`reconnects_total`	ressurreições de um pusher morto	cresce com `status: running` — o canal rompe e se recupera
`retransmitted_packets_total`	retransmissões do transporte (SRT, RIST)	crescem — perdas no caminho até o receptor
`datagrams_total`, `frames_total`	as unidades enviadas: datagramas nos transportes de pacotes, quadros no RTMP	—

Campos zerados são omitidos da resposta: um push RTMP não tem datagramas, um UDP não tem quadros nem retransmissões.

Ao lado dos contadores do stream, como campos de nível superior da mesma resposta, há três somas sobre todos os pushes — `push_bytes_total`, `push_errors_total`, `push_reconnects_total`. Elas respondem «quanto o stream enviou no total» sem percorrer o mapa.

SÉRIES DO PROMETHEUS

Cada contador de push é uma série própria com o rótulo `push`:

Série	Rótulos
`stream_push_bytes_total`	`stream`, `push`, `proto`
`stream_push_errors_total`	`stream`, `push`, `proto`
`stream_push_reconnects_total`	`stream`, `push`, `proto`
`stream_push_retransmitted_packets_total`	`stream`, `push`, `proto`
`stream_push_datagrams_total`, `stream_push_frames_total`	`stream`, `push`, `proto`

No servidor Prometheus embutido os mesmos contadores estão disponíveis com os rótulos `name` (o stream) e `push`; no central soma-se `node` a eles. A profundidade de protocolo — `stream_push_srt_*` e `stream_push_rist_*` — chega no scrape do nó junto com a opção de contadores estendidos.

```
# velocidade de envio por destino
rate(stream_push_bytes_total{stream="tv1"}[1m])

# a saída total do stream por todos os pushes
sum by (name) (rate(stream_push_bytes_total{name="tv1"}[1m]))

# canais que rompem: reconexões em cinco minutos
```

```
increase(stream_push_reconnects_total[5m]) > 0

# perdas no caminho até o receptor por SRT/RIST
rate(stream_push_retransmitted_packets_total[1m])
```

COMO LER O ESTADO

- `status: error` **com** `error` **não vazio** — nada está sendo enviado, e a causa está nomeada. O push continua se levantando sozinho, uma vez a cada cinco segundos.
- `status: running`**, mas** `bitrate_kbit` **zerado** — a conexão está lá, os dados não: confira se o próprio stream está rodando.
- `status: running` **com** `reconnects_total` **crescente** — o canal rompe e se recupera; olhe `opened_at`, ele mostra a idade da conexão atual.
- `status: disabled` — o push está desligado na configuração. Seus contadores ficam congelados nos últimos valores; nada é perdido.

O servidor Prometheus embutido

O Sapsan contém um servidor Prometheus embutido: a HTTP API `/api/v1/query`, `/api/v1/query_range`, `/api/v1/series`, `/api/v1/labels`, `/api/v1/label/{name}/values`, `/api/v1/status/buildinfo` sobre um armazenamento próprio em memória. Para o Grafana é um datasource pronto: adicione um datasource do tipo Prometheus, aponte para a URL do servidor — e construa gráficos sem implantar uma TSDB. Os gráficos da UI embutida se alimentam dessas mesmas consultas.

Limites de construção:

- o histórico é de algumas horas, em memória; depois de um restart o armazenamento fica vazio. É o canal do «o que está acontecendo agora», não um arquivo de métricas;
- o PromQL é suportado como um subconjunto: seletores com matchers de rótulos, `rate` / `irate` / `increase`, as agregações `sum` / `avg` / `min` / `max` / `count` com `by()` / `without()`, aritmética, `offset`. Uma construção não suportada devolve um erro explícito, não um resultado distorcido;
- o conjunto de séries é o básico; as séries profundas (por PID, SRT, RTP) aparecem com a opção de contadores estendidos.

CONSULTAS A UM ÚNICO SERVIDOR

```
# velocidade de captura do stream, bytes/s
rate(stream_input_bytes_total{stream="cam1"}[1m])

# erros de entrada nos últimos 5 minutos
increase(stream_input_errors_total{stream="cam1"}[5m])

# o arquivo está sendo escrito com erros?
increase(stream_dvr_write_segments_failed_total{stream="cam1"}[10m])

# tráfego total de captura do servidor
sum(rate(stream_input_bytes_total[1m]))

# memória usada pelos streams, por parte do pipeline
sum by (part) (stream_memory_bytes)
```

Num servidor único as séries não carregam o rótulo `node`.

CONSULTAS AO CENTRAL

O central e os nós Sapsan formam um único complexo: o central coleta as estatísticas de todos os nós, e o servidor Prometheus embutido dele serve exatamente o mesmo API. Há uma única diferença, mas ela atravessa tudo — o **rótulo** `node`:

- cada série carrega `node="nome-do-nó"` — dá para ver qual nó a produziu;
- um stream pode dar várias séries: um stream de cluster é servido em partes em nós diferentes;
- `node` aparece em `/api/v1/labels` e em `label_values(node)` — é disso que se faz uma variável de dashboard do Grafana com um menu suspenso de nós.

```
# o stream inteiro, não importa em quantos nós ele vive
sum without (node) (rate(stream_input_bytes_total{stream="cam1"}[1m]))

# tráfego de captura do complexo, detalhado por nós
sum by (node) (rate(stream_input_bytes_total[1m]))

# em quais nós o stream vive agora — olhe o rótulo node do resultado
stream_input_bytes_total{stream="cam1"}
```

O mesmo dashboard funciona tanto contra um Sapsan único quanto contra o central: agregue com `sum without (node)` — num servidor único o rótulo não existe, e a agregação não muda nada.

O scrape do Prometheus: o seu próprio monitoramento

> **A opção «contadores estendidos»**
>
> Os endpoints do scrape estão disponíveis só com a opção de licença de contadores estendidos. Sem ela, use o servidor Prometheus embutido e o Retroview.

Para instalações com infraestrutura de monitoramento própria, o Sapsan serve métricas no formato de texto do Prometheus:

- `GET /streamer/api-v4/live-metrics` — streams e entradas;
- `GET /streamer/api-v4/sessions-metrics` — sessões de reprodução;
- `GET /streamer/api-v4/dvr/metrics` — discos e catálogo do arquivo;
- `GET /streamer/api-v4/runtime/metrics` — o processo e o alocador.

Diferenças em relação ao servidor Prometheus embutido: o histórico fica guardado do seu lado e é limitado só pelo retention da sua TSDB; a profundidade completa está disponível, incluindo as métricas de protocolo por PID/SRT/RTP; o alerting é o seu alertmanager. Os contadores são monótonos e sobrevivem à observação de restarts — este é também o canal para a conciliação de faturamento do seu lado.

Contadores estendidos

A opção de contadores estendidos abre o detalhamento causal e de protocolo em todo lugar ao mesmo tempo: nas seções `input` e `dvr.write` da Admin API, nas séries do servidor Prometheus embutido e no scrape do Prometheus. Na telemetria do Retroview esses contadores sempre vão, independentemente da opção — por isso a análise de causas está disponível no Retroview mesmo sem ela.

CAUSAS DOS ERROS DE ENTRADA

O `errors` agregado responde «há um problema na entrada»; o detalhamento causal responde «qual exatamente». As causas vêm em duas famílias:

- **defeitos de mídia de um stream vivo** — `lost_packets`, `broken_payload`, `desync`, `ts_pat`: os dados chegam, mas danificados;
- **falhas da fonte** — `connection_closed`, `connection_refused`, `timeout`, `http_request_error`, `not_found`, `denied`, `decode_error`, `protocol_error`, `io_error`, `other`: a fonte morreu, e a causa da morte está classificada.

A classificação vem da categoria tipada do erro, e não do texto do log, e cobre inclusive a falha ao abrir uma fonte. Cada falha entra também no `errors` agregado; as falhas repetidas nas tentativas são contadas a cada vez — a frequência de eventos é exatamente o sinal de «a fonte ainda está morta». As paradas normais — reconfiguração, fim do stream, desalojo por uma fonte de maior prioridade — não contam como erros.

A série do detalhamento

No servidor Prometheus embutido o detalhamento viaja como uma série com o rótulo de causa — `errors_detail_total{name, cause}`; no central as séries levam também `node`. Causas sem eventos não criam séries.

```
# erros do stream em 5 minutos, detalhados por causa
sum by (cause) (increase(errors_detail_total{name="tv1"}[5m]))
```

MPEG-TS POR CADA PID

Por cada PID do fluxo de transporte: `errors_ts_cc` (violações de continuity counter — o principal indicador de perdas de transporte), `errors_ts_tei`, `errors_ts_scrambled` (a criptografia não foi retirada — problemas de CAM/chaves), `errors_ts_psi_checksum`, PES quebrados, buckets de jitter do PCR, esvaziamentos do buffer do decodificador (HRD).

Por quê: este é material de monitoramento de nível broadcast. Um alerta sobre `rate(errors_ts_cc) > 0` por PID pega a degradação do transporte antes que o espectador a veja; o jitter do PCR e o HRD mostram se o stream sobreviverá a um decodificador de hardware.

SRT

RTT e sua variação, perdas e retransmissões nos dois sentidos, descartes por atraso, enchimento dos buffers, tempos limite de keepalive.

Por quê: ajustar a `latency` ao canal real e diagnosticar «quem perde — nós ou o lado remoto». As retransmissões crescem com o RTT estável — perdas no caminho; o RTT pula — o canal está congestionado.

RTP/RTSP POR CANAL

Pacotes perdidos, saltos e travamentos dos timestamps, NACKs, transbordamentos de buffer — por cada canal RTP (vídeo/áudio) de uma câmera.

Por quê: diagnóstico do parque de câmeras — qual câmera está falhando, antes de chegarem as reclamações de artefatos.

DESEMPENHO DE ESCRITA DO DVR

`segments_written_slow`, `segments_written_delayed`, `segments_written_collapsed`, `segments_discontinuity`.

Por quê: um sinal precoce de «o disco não dá conta». A fatia de slow/delayed cresce com o mesmo tráfego — o armazenamento está degradando; `discontinuity` significa buracos que você verá depois em `recorded_hours`.

Retroview

O Retroview é o monitoramento em nuvem do fabricante. Uma vez por minuto o Sapsan envia telemetria — o catálogo completo de contadores, incluindo todo o detalhamento estendido — e o Retroview o guarda por meses. No servidor não há nada para configurar: o canal funciona junto com a licença.

Quando ir ao Retroview, e não ao servidor Prometheus embutido:

- **histórico e tendências** — o crescimento do tráfego e do número de streams ao longo de meses, planejamento de capacidade;
- **post-mortems** — o que estava acontecendo com o stream de madrugada: a telemetria sobrevive a restarts e não depende da memória do servidor;
- **conciliação de consumo** — os contadores são amostrados no tempo por um sistema externo, então a conta pode ser conferida mesmo que os servidores tenham sido reiniciados;
- **análise de causas sem a opção de contadores estendidos** — a telemetria sempre carrega o detalhamento causal;
- **alertas prontos** — regras de produto em vez de regras caseiras.

Receitas

Pergunta	Onde olhar	O sinal do problema
O stream está vivo?	`status` na API; `rate(stream_input_bytes_total[1m])` no Prometheus embutido	status não é `running`; a velocidade de captura despencou para zero
A reserva está pronta?	`inputs[]` na lista de estatísticas	`status` é `error` ou `unchecked`; `checked_ago_ms` muito acima do intervalo de verificação
Estamos vivendo na reserva?	`source.url`; `num_sec_on_secondary_input`	a URL não é a primeira do config; os segundos na reserva seguem crescendo
A entrada está degradando?	`input.errors`, `input.retries`; as causas — contadores estendidos ou Retroview	os contadores crescem com o stream vivo
O push chega?	`pushes[].status` e `bitrate_kbit`; `rate(stream_push_bytes_total[1m])`	status não é `running`; velocidade zerada com o stream vivo
Por que o push não funciona?	`pushes[].error`	a causa em palavras: conexão recusada, tempo limite, recusa de publicação
O canal até o receptor rompe?	`pushes[].reconnects_total`, `opened_at`	as reconexões crescem; `opened_at` fica mais novo a cada poucos minutos
O arquivo está sendo escrito?	`dvr.to` na lista; `dvr.write.segments_failed` no GET individual	`to` fica atrás da hora real; `segments_failed` cresce
Há buracos no arquivo?	`dvr.recorded_hours` contra a janela `to - from`	as horas gravadas são notavelmente menos que a largura da janela
Os discos dão conta?	`segments_written_slow/_delayed` (opção); `dvr_disk_free_bytes` no scrape	a fatia de escritas lentas cresce; o espaço livre derrete mais rápido que o esperado
O servidor está saudável?	`runtime/metrics`: `process_rss_bytes`; o Prometheus embutido: `stream_memory_bytes`	a memória cresce monotonicamente sem crescimento de carga
O que aconteceu de madrugada?	Retroview	—

3.7.5 Monitoramento

O Sapsan escreve logs estruturados, serve métricas em formato Prometheus e exporta traces por OpenTelemetry (OTLP) — se encaixa no stack padrão de observabilidade: Prometheus/Grafana, Jaeger/Tempo, Loki.

Logs

O nível de log é controlado pela variável de ambiente `RUST_LOG`:

```
RUST_LOG=info sapsan
RUST_LOG=debug,hls=trace sapsan   # mais detalhe para um módulo específico
```

> **Note**
>
> TODO: formato dos logs, destinos padrão nas instalações deb/Docker.

Métricas Prometheus

`GET /streamer/api-v4/runtime/metrics` serve métricas do processo em formato Prometheus (incluindo o alocador jemalloc). Ao lado estão as métricas de live, de sessões e de DVR: `/live-metrics`, `/sessions-metrics`, `/dvr/metrics` (veja o Admin API). Os endpoints do scrape exigem a opção de licença de contadores estendidos; sem ela, os gráficos são servidos pelo servidor Prometheus embutido (um datasource Grafana pronto). O percurso completo pelos canais de estatística, os campos e as receitas de alertas está na página Estatísticas.

Contadores de qualidade

O que está disponível para alertar:

- **Captura MPEG-TS**: por cada PID — erros CC, TEI, pacotes cifrados, erros CRC de PSI, PES quebrados, estado do buffer do decodificador (HRD).
- **SRT**: RTT, perdas e retransmissões nos dois sentidos, tamanhos de buffer, tempos limite de keepalive.
- **Fontes (LSI)**: tempo na entrada principal/reserva, tempo sem dados, tentativas, validade das reservas, divergência de parâmetros entre as entradas.
- **Sessões**: abertas, recusadas, autorizadas, recusas de reautorização.
- **DVR**: espaço em disco, contadores de operações, número de blobs, bytes usados pelo arquivo, progresso da reconstrução do catálogo.
- **Pushes**: por push — status com a causa da falha, velocidade de envio, bytes, erros, reconexões, retransmissões SRT/RIST. Veja Estatísticas de pushes.
- **Agentes do Peeklio**: estado das conexões, bytes, erros.

Traces

O Sapsan exporta traces pelo protocolo OTLP.

> **Note**
>
> TODO: variáveis de ambiente do exportador OTLP (endpoint, sampling), o que os traces cobrem.

Saúde do servidor

- Sonda de readiness: `GET /streamer/api-v3/monitoring/readiness` — para Kubernetes e balanceadores.
- Estado dos streams: `GET /streamer/api-v3/streams`.

Ferramentas de diagnóstico

Os validadores HLS e DASH embutidos verificam de forma ativa a entrega própria (ou a alheia): latência LL-HLS, integridade dos timelines, alinhamento da escada ABR.

3.7.6 Licenciamento

O Sapsan é licenciado por meio de um arquivo de licença emitido ao cliente no momento da compra. Você mesmo pode baixar a chave de licença na área do cliente em my.flussonic.com.

A licença é verificada online: o servidor precisa de acesso à internet aos servidores de licenciamento da Flussonic. Sem acesso à internet o Sapsan não funciona.

Os servidores de licenciamento da Flussonic estão distribuídos por continentes e regiões — a falha de um único servidor ou a indisponibilidade de uma região inteira não afeta a verificação da licença.

> **Note**
>
> O Sapsan nunca se atualiza sozinho. Não existe funcionalidade de atualização forçada, e nunca vai existir: a verificação da licença não altera o software instalado, e subir de versão é sempre uma ação explícita do administrador por meio do gerenciador de pacotes.

Arquivos

A licença é um único arquivo que você coloca junto ao config; o resto dos arquivos o próprio Sapsan cria no diretório de estado durante a ativação.

Arquivo	Localização	Conteúdo
`license.txt`	`/etc/sapsan/` (junto ao `sapsan.yaml`)	a chave de licença emitida na compra
`server.id`	`/var/lib/sapsan/`	identificador único do servidor (UUID)
`activation-<version>.json`	`/var/lib/sapsan/`	ativação para uma versão específica do produto
`keys/`	`/var/lib/sapsan/`	diretório com o material de chaves gerado pelo Sapsan

As variáveis de ambiente controlam onde o Sapsan procura os arquivos:

- Diretório de estado — `SAPSAN_STATE_DIR`; em caso contrário, o diretório pai de `SERVER_ID_PATH`; em caso contrário, o diretório de trabalho atual. O pacote deb e a imagem de Docker definem `SERVER_ID_PATH=/var/lib/sapsan/server.id`, portanto o diretório de estado é `/var/lib/sapsan`.
- Arquivo de licença — `LICENSE_PATH`; em caso contrário, `{dirname(CONFIG_PATH)}/license.txt`; em caso contrário, `{state_dir}/license.txt`. Com `CONFIG_PATH=/etc/sapsan/sapsan.yaml` isso é `/etc/sapsan/license.txt`.

Como colocar a licença antes do primeiro início — veja instalação e execução.

Estados

Estado	Quando	O que funciona
`Licensed`	a licença está verificada	tudo
`Restricted`	a licença está ausente, expirou ou não passou na verificação	a interface de administração continua disponível, o streaming fica limitado

O status atual da licença é entregue pela API: `GET /streamer/api-v4/license/status`.

O que vem a seguir

- Admin API

3.7.7 Compatibilidade com o Flussonic

O Sapsan responde nas URLs habituais do Flussonic Media Server para que os reprodutores, as integrações e o monitoramento existentes continuem funcionando durante a migração. A compatibilidade vem ligada por padrão e pode ser desligada no config:

```
flussonic:
  streaming: false   # por padrão true
```

URLs legadas suportadas

São reconhecidos endereços de nível raiz dos seguintes formatos (apenas GET/HEAD/OPTIONS):

URL legada	O que faz
`/<stream>/variant/v1/index.m3u8`	playlist da faixa
`/<stream>/info.json`	informação do stream
`/<stream>/ranges.json`	intervalos gravados do arquivo
`/<stream>/<utc>-preview.mp4`	preview MP4 de um momento do arquivo
`/<stream>/index-<from>-<duration>.fmp4.m3u8`	playlist HLS do arquivo; redireciona (302) para `/streaming/v/<stream>/index.m3u8?from=&to=`, convertendo segundos em milissegundos

Parâmetros de `ranges.json`: `closed_at_gte`, `opened_at_gte`, `opened_at_lte`, `resolution`, `limit`, `cursor`.

Admin API v3

A API sob o prefixo `/streamer/api/v3` responde no formato do Flussonic Streamer API v3 — veja Admin API. Na conversão de uma configuração v3, alguns campos são descartados de propósito (`static`, `comment`, `title`, `segment_duration`, listeners `https` e outros) — o relatório de perdas fica disponível durante a conversão.

Tokens

Como no Flussonic, o token é aceito tanto como `?token=` na URL quanto no cabeçalho `Authorization: Bearer` (o cabeçalho tem prioridade).

O que vem a seguir

- Admin API
- Autorização