Software Origin Use Cases

Document no	EDGS-216
Version	A19
Created on	2026-06-10

Contents

1 Introduction

1.1 Confidentiality Notice

This document is confidential and may not be reproduced, distributed or used for any purpose other than by the recipient for the assessment, evaluation and use of AgileTV products, unless written permission is given in advance by AgileTV.

1.2 About this Document

This document describes how to set up a Software Origin System consisting of Convoy, SW Live Ingest, the Recorder/VoD Ingest tool, SW Repackager as well as Orbit TV Server and Software Streamer.

1.3 References

[1] EDGS-069 - Convoy Management Software - User Guide

[2] EDGS-122 - Convoy Management Software - CCMI Specification

[3] EDGS-187 - ESB3003 Live Ingest User Guide

[4] Nginx documentation - http://nginx.org/en/docs/

[5] Timed Text Markup Language 1 (TTML1) - https://www.w3.org/TR/ttaf1-dfxp/

[6] DASH Industry Forum Profiles - http://dashif.org/identifiers/profiles/

[7] EDGS-156 - TV content capture - CCMI CLI

[8] EDGS-176 - SW Repackager User Guide

[9] EDGS-208 - ESB3005 User Guide

[10] EDGS-162 - Preferred Format for External ABR VoD Libraries

[11] EDGS-168 - DRM Gateway User Guide

[12] EDGS-184 - Live and Catchup with TV Content Capture and TV Repackaging

[13] Low Latency DASH - https://dashif.org/guidelines/iop-v5/#part-4-live-and-low-latency-services

[14] Low Latency HLS - https://developer.apple.com/documentation/http-live-streaming/enabling-low-latency-http-live-streaming-hls

1.4 History

Version	Date	Changes
A1	2018-12-13	This is the initial version of the document
A14	2019-11-13	Added use case for VoD ingest to AWS S3 object storage
A15	2019-12-13	Added use case for VoD ingest to Azure blob storage. Added use cases for CBM recordings for AWS and Azure object storage.
A17	2022-02-22	Drop unmanaged VOD support
A18	2024-10-15	Added use case for Low Latency Live Streaming
A19	2024-11-06	Update about Low Latency HLS

1.5 Third party acknowledgement

The Software Repackager uses the Bento4 C++ library to read MP4 files in some workflows.

2 The Software Origin System

2.1 Overview

The Software Origin System makes it possible to:

- Run a service with Live TV channels.
- Support catchup TV.
- Make recordings from catchup buffers connected to live channels.
- Ingest, repackage and stream VoD assets on the fly.
- Convert assets that are stored in VoD libraries into a common storage format for subsequent repackaging and streaming.

The common storage format that is used by the Software Origin System consists of MPEG CMAF (Common Media Application Format) files for audio, video, and text data, in combination with a number of additional metadata files. This complete set of files is referred to as a managed asset or an ESF (Edgeware Storage Format) asset.

2.2 System components and ESB numbers

The system components are released as ESBs. The table below summarizes the mappings between those.

ESB	System Component Name
3003	SW Live Ingest
3002	SW Repackager
3005	Recorder or ew-vod-ingest
3009	DRM GW
2001	Orbit TV Server
3006	Convoy

2.3 Useful Tools

2.3.1 Repackager Tool

The `ew-repackager-tool` that is installed along with the Software Repackager is a utility tool that allows the user to test parts of the functionality inside the Software Repackager from the command line.

Please read the Tools section in [8] for details on how to run `ew-repackager-tool` to e.g. validate managed VoD content or issue stream requests locally on the repackager node.

2.3.2 Live Ingest Tools

See the SW Live Ingest ESB3003 User Guide to learn about the tools available:

- `ew-liveingest-tool` to verify the status of live channels.
- `ew-cb-media` to verify the corresponding catchup buffers.
- `ew-diagnose-tool` to troubleshoot a channel.

2.3.3 Recorder Utilities

The installation of ESB3005 will provide a number of utilities that can be used to investigate and manipulate recordings done with the WebTV Recorder in ESF format (CMAF + metadata). See [9] for more information.

3 General Notes for Installation and Setup

3.1 Convoy

Refer to the Convoy User Guide [1] for information on how to install Convoy.

NOTE: Example uses of the `channel` and `content` command-line tools in this document do not supply the `-k` option, and therefore assume that the `ACCT_CLI_APIKEY` environment variable has been set on the `mgmt` machine where they are run:

```
$ export ACCT_CLI_APIKEY=$(convoy-admin show api-key)
```

3.2 ESB3005

The ESB3005 software bundle contains tools to perform recordings and VoD ingest. The ESB3005 is provided on the Convoy ISO as a convenience but can be upgraded separately.

The Recorder is installed together with the recording agent automatically when Convoy is installed. The Recorder will be installed to `/usr/bin/ew-recorder`. It will be invoked as required by Convoy, and log output from the Recorder can be viewed with `journalctl -u convoy-recording-agent`.

For information on how to install or upgrade the Recorder, consult the ESB3005 User Guide listed in [9].

3.3 ESB3009 DRM Gateway

The DRM (edrm) Gateway is delivered as a separate software bundle called ESB3009. It comes together with a user manual describing how to install or upgrade it. It is recommended to install edrm on repackaging nodes.

3.4 Software Repackager

Please see [8] for details on how to install the Software Repackager.

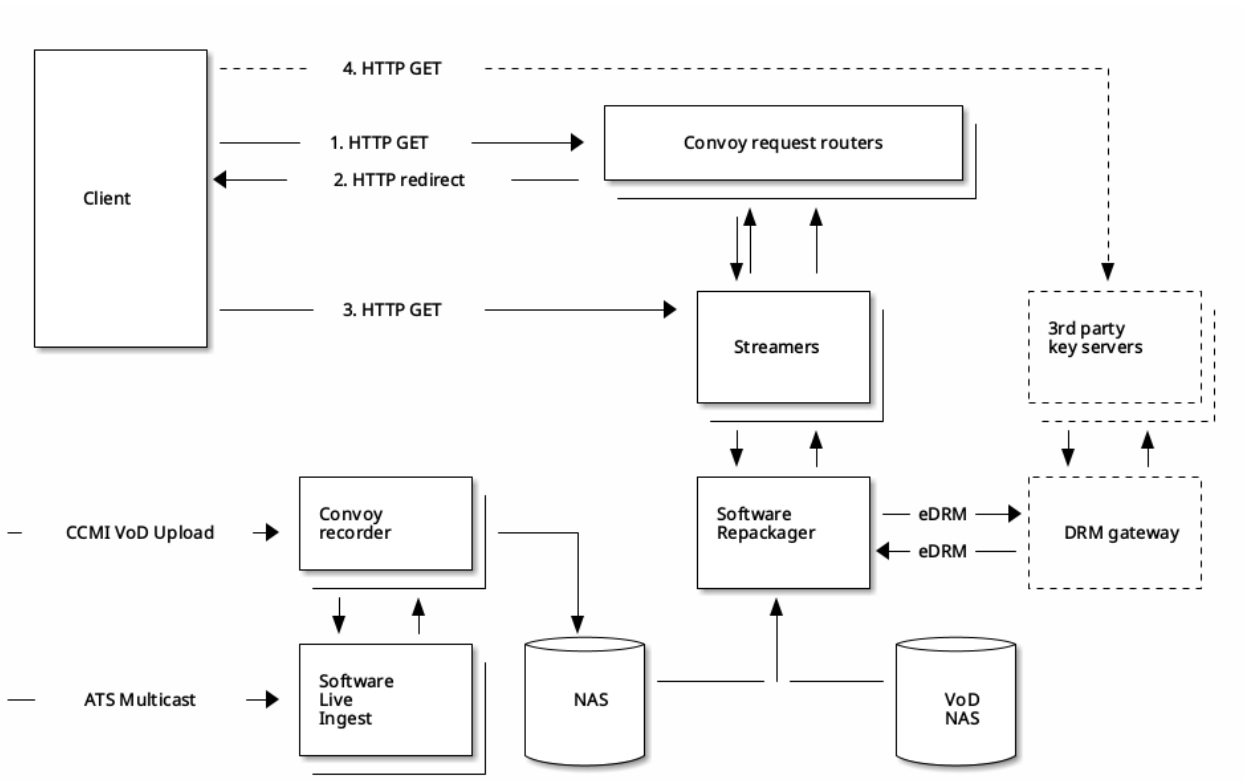
The Software Repackager will typically run on dedicated machines. In order to make those machines visible in the Convoy web UI, include them in the `repack` role in `install.conf`. Example:

```
hosts {
    ...
    repack = { repack-1 repack-2 }
}
```

Note that in order to expose the Software Repackager the firewall needs to be configured to open port 80 on the repack nodes. Consult the Convoy User Guide sections 3.6.10 Configure the Firewall and B.2 Configuring iptables for more information.

4 Use Case: Recordings

When recording from live channels, all recordings are made in the past in the sense that they will not start before the end-time of the recorded program has passed. Thus they require a live buffer to be configured and used. By default, the recording will be made one hour after the program end time. An advantage of making the recordings in the past is that it is possible to reschedule the recording if the program start or end time has changed, e.g. due to an extended sports event.



A typical installation will run the Convoy recorder on the same machine as SW Live Ingest. The SW Repackager will typically run on a dedicated machine.

4.1 Prerequisites

The table below summarizes the software versions used for this use case description.

System Component	Release version
ESB3003 SW Live Ingest	1.10
ESB3002 SW Repackager	1.14
ESB3005 Recorder/VoD Ingest	2.18
ESB2001 Orbit TV Server	Any
Convoy	2.12

4.2 Installation

Perform the installation according to the section [General Notes for Installation and Setup]. Also note the following:

- Depending on the preferred deployment model, the file copies required for the recordings can either be performed on a Convoy mgmt machine, a dedicated recorder machine in the Convoy cluster, or on a SW Live Ingest machine.
- The recorder must run on the same machines that run Convoy's `recording-agent` service. By default, this means the machines in the `mgmt` role. To run the recorders on dedicated machines or the SW Live Ingest machine, simply specify the `recording-agent` role in Convoy's `install.conf`. For example (using hostnames `live-ingest-1` and `live-ingest-2`):

```
hosts {
    ...
    recording-agent = { live-ingest-1 live-ingest-2 }
}
```

4.3 Configuration

4.3.1 SW Live Ingest

In order to be used as a source for the Recorder, the `ew-cbm` service must be exposed to the Recorder. If the Recorder (and recording agent) runs on another machine, make sure the firewall is opened.

In addition, any channel to be recorded must exist in the SW Live Ingest channel configuration on the server, and have a circular buffer sufficiently large to hold the longest recording plus the `buffered_schedule_delay`. For configuration of the `buffered_schedule_delay` see the section "Scheduling Delay".

4.3.2 Software Repackager

In order to stream content that has been ingested by the Recorder, content locations need to be configured as sources in the Software Repackager.

Please see the Configuration section in [8] for details on how to configure sources in the Software Repackager.

4.3.3 Convoy

4.3.3.1 Repackagers, Live Ingest Servers and Storage Systems

Follow the section *Set Up CCMI WebTV Recordings* in the Convoy user guide [1]. In summary:

- Add repackagers as origin servers. This is required by Convoy so there is a link between the origin and the CDN.
- Add live ingest nodes as `webtv_recorders`.
- Specify buffer size and `ingest_script`.

Take special care to configure the storages in the correct way. In Convoy you configure the *target* storage, where finished recordings will end up. This storage can be either a mounted file system or an FTP host. In addition to this, you must also make sure the catchup buffers are mounted on the machine running the recording agent service. The mount point must be *exactly* the same as it is on SW Live Ingest. The reason is that the CBM service in SW Live Ingest provides absolute paths to the recording agent that shall be copied. This source storage is not present in the Convoy configuration.

Note that the repackager needs to be configured to read the same storage root, but may have a different path to access it. Please see the Configuration section in [8] for details on how to configure managed sources in the Software Repackager.

4.3.3.2 Edge Caches

The edge caches are listed under `streamers` in `convoy.conf`. They must be manually configured with the proper backends to be able to find the repackagers. The recommended way to do this is to use the `auto` streamer configuration mode, and to specify the necessary backend using the `extra_config` mechanism.

Using the example above, a backend with prefix `/__cl/s:storage-1,storage-2` should be created, and it should fetch content from `repack-1` (primary) and `repack-2` (secondary). In `auto` mode, Convoy will automatically create streamer groups named `origin_server:repack-1` and `origin_server:repack-2`, so these can be used in the backend configuration. Create a backend on the streamer using e.g. `confcli services.webtv.origin.backends -w`. The result should look like:

```
"backends": [
    {
        "authorization": {
            "enable": false,
            "password": "",
            "user": ""
        },
        "caseSensitive": true,
        "prefix": "/<prefix>/__cl/s:storage-1,storage-2",
        "name": "jitp",
        "rewrite": {
            "that": "",
            "this": "/<prefix>"
        },
        "serverGroups": [
```

```
            "origin_server:repack-1",
            "origin_server:repack-2"
        ]
}
```

The prefix <prefix> must be substituted with your account prefix. Run `convoy-admin account list --verbose` to list account prefixes.

Refer to the Convoy User Guide for information on how to add this file to the streamer configuration using the `extra_config` property.

Note that streaming interfaces and session support must also be enabled using `extra_config` for automatic streamer configuration to work as expected (this requirement will be removed in later versions of Convoy). Depending on if the version of the streamer used supports IPv6 or not, care has to be taken in order to configure the streaming interfaces correctly with `extra_config` parameter by either defining the `ipAddress6` key or not.

If two edge caches are used, it is recommended to swap the order of the primary and secondary repackaging node on the second edge, so that at least one edge can still reach its primary if one of the repackaging nodes should go down.

The resulting streamer configuration in `convoy.conf` will look something like this:

```
streamers {
  edge-1 {
    host = 10.16.1.160
    parent = edge-1
    configuration = auto
    extra_config = {
      @/etc/streamer_conf/repack_backend_1_2.json
      @/etc/streamer_conf/streaming_interfaces.json
      @/etc/streamer_conf/sessions.json
    }
  }
  edge-2 {
    host = 10.16.1.161
    parent = edge-2
    configuration = auto
    extra_config = {
      @/etc/streamer_conf/repack_backend_2_1.json
      @/etc/streamer_conf/streaming_interfaces.json
      @/etc/streamer_conf/sessions.json
    }
  }
}
```

Note that it is not a requirement to use the `origin_server:*` server groups defined by Convoy in the backend configuration. If specialized behaviour is needed, just add the necessary server groups using `extra_config` and refer to those in the backend instead.

4.3.3.3 CCMI Channels

Recordings are scheduled in Convoy via the CCMI interface. For that to be possible, Convoy must first be made aware of the channels that can be recorded. This can be done by using the `channel` CCMI CLI wrapper on any Convoy `mgmt` machine:

```
[root@mgmt-1 ~]# channel add -t "Edgy TV" edgytv -T mbr_ts_multicast
```

The channel name "edgytv" above must match the name of the corresponding channel in SW Live Ingest. The `mbr_ts_multicast` parameter must be specified to indicate that this channel will be used for Web TV recordings.

4.4 Testing the Installation

In order to verify that the system has been set up correctly, the following three steps are suggested. Consider enabling debug logging in Convoy before starting to make it easier to follow what is going on.

In production, recordings will typically be scheduled via the CCMI HTTP API (see the CCMI Specification for details), but for testing purposes it is convenient to use the command-line CCMI wrappers that Convoy provides, i.e. the `channel` and `content` commands.

4.4.1 Schedule and perform a recording

Schedule a recording from the channel `edgytv` using `content record`:

```
[root@mgmt-1 ~] content record -c edgytv -S HTTP -t "Evening News" \
  --start "2016-06-01T18:00:00Z" --stop "2016-06-01T19:00:00Z" evening-news
"evening-news" will record from 2016-06-01T18:00:00Z to 2016-06-01T19:00:00Z
   ↪ on channel "edgytv"
```

The recording will start once the stop time plus the `ccmi.webtv_recording.buffered_schedule_delay` defined in `convoy.conf` has passed. Check progress with `journalctl -u convoy-recording-agent` on the `recording ↪ -agent` nodes.

If there are multiple `webtv_recorder` entries in `convoy.conf`, one recording will be performed from each. The jobs will be spread randomly across the available recording agents, and it is possible that the same recording agent will perform several instances of the same recording.

If the scheduling fails, check for errors with:

```
    jorunalctl -u convoy-acctapi
```

and

```
    jorunalctl -u convoy-rec2
```

The `content` command can also be used to cancel a recording, or reschedule a pending recording. Run `content --help` and `content <CMD> --help` for more details.

The finished recording will end up on the storages defined in `convoy.conf` in a structure similar to this:

```
/mnt/nas2/rec-root/r/5/e/6/
|
|-- r_eveningnews_5e6cd2bd6d3a72e1723c288234670da6
|   |-- content_info.json
    |-- video_1.cmfv
    |-- video_1.dat
    |-- video_2.cmfv
    |-- video_2.dat
    |-- audio_eng_0.cmfa
    |-- audio_eng_0.dat
    |-- audio_chi_0.cmfa
    |-- audio_chi_0.dat
    |-- sub_eng_0.cmft
    |-- sub_eng_0.dat
    |-- sub_eng_0.cmft
    |-- sub_eng_0.dat
```

Each recording will be stored in a separate folder and should have one `content_info.json` file describing the content, a `[variant].dat` file describing the segments for each variant and a `[variant].cmf[x]` file, where `x` is `v`, `a`, or `t` for video, audio, and text, respectively. The file extensions and the variant containers follow the proposed MPEG CMAF format which uses the MPEG-4 ISOBMFF file format.

It is possible to play a ".cmfv" file using an application like VLC to verify that the recording has the expected duration and content.

4.4.2 Serve recording directly from Software Repackager

Once a recording exists on the shared storage, the next logical step is to test the repackager by requesting an HLS version of the recording directly from Nginx.

Use `content get` to find the HLS delivery URI for the recording (URIs somewhat shortened for easier presentation below):

```
[root@mgmt-1 ~]# content get evening-news
title:    Evening News
...
delivery URIs:
  hls:     /__cl/s:storage-1,storage-2/__c/r/6/1/3/r_eveningnews_613...bac/
     ↪ __op/hls-default/__f/index.m3u8
  mss:     /__cl/s:storage-1,storage-2/__c/r/6/1/3/r_eveningnews_613...bac/
     ↪ __op/mss-default/__f/Manifest
  dash:    /__cl/s:storage-1,storage-2/__c/r/6/1/3/r_eveningnews_613...bac/
     ↪ __op/dash-default/__f/manifest.mpd
```

The HLS master playlist can then be fetched using `curl` like this:

```
$ curl http://repack-1/__cl/s:storage-1,storage-2/__c/r/6/1/3/
   ↪ r_eveningnews_613...bac/__op/hls-default/__f/index.m3u8
```

If the above command fails, check the repackager's configuration and logs.

To actually play back the recording, simply point the client to the above URL.

4.4.3 Serve recording via request router and edge cache

When both the Recorder and the Software Repackager have been verified to work correctly, the final step is to bring Convoy's request router and the edge caches into the mix.

The only change compared to above is that the request should be made against the Convoy request router rather than against the repackager node. By also adding the `-L` switch to `curl`, the redirect from the request router will be followed. This should result in the edge serving the master playlist. Assuming there is a request router running on machine `rr-1`:

```
$ curl -L http://rr-1/__cl/s:storage-1,storage-2/__c/r/6/1/3/
   ↪ r_eveningnews_613...bac/__op/hls-default/__f/index.m3u8
```

If the request router fails to respond with a 302 redirect to a streamer, check the logs by:

```
    journalctl -u esb3008@*
```

If the streamer returns an error when the redirect is followed, check the logs on the Orbit.

5 Use Case: Upload VoD Content using the CMS API (CCMI)

CCMI is the name of CMS API and it allows administrators to upload VoD content from different sources and formats and store them in a single format, ESF. Content stored in the ESF format can then be processed by Software Repackager to serve different types of clients. The following types of media source formats are supported:

- HTTP Live Streaming (HLS) (Both Byte-Range and file segments)
- Microsoft Smooth Streaming (MSS) Client Format
- MPEG-DASH on-demand
- MP4 (SMIL)

Each source can be retrieved from the filesystem or from a remote location using HTTP or FTP.

5.1 Supported Media Formats

The following media formats are supported when uploading VoD Content to ESF:

- Video
  - H264
  - HEVC
- Audio
  - AAC
- Subtitles
  - TTML (MP4, MPEG-DASH on-demand, and MSS Source)
  - WebVTT (MPEG-DASH on-demand and HLS Source)

When uploading a non-segmented media source like MPEG-DASH on-demand or MP4 (SMIL) content, the segmentation logic requires that there are common synchronization points among the different video tracks. This, more or less implies that the source content also must have all its video tracks in the same timescale in order to be successfully segmented.

The only TTML input that is supported for MPEG-DASH on-demand and MP4 VoD media sources, is side-car XML files. I.e. not segmented subtitles. The mime-type must be application/text+ttml.

For DASH on-demand, side-car WebVTT files are supported. They have mime-type 'text/vtt'.

5.1.1 TTML Format Details

Subtitles are stored in TTML (Timed Text Markup Language) as defined in [5]. For each language there is a separate `div` tag in which all the cues for that language are listed. The language of each subtitle track is specified with the `xml:lang` attribute within the `div` tag:

```
<tt>
  ...
  <body>
    <div xml:id="captions" xml:lang="ger">
      <p begin="00:00:10:920" end="00:00:11:360">Ein Paar Schuhe, Bitte.</p>
      ...
    </div>
    <div xml:id="captions" xml:lang="ita">
      <p begin="00:00:10:920" end="00:00:11:360">Un paio di scarpe, per
         ↪ favore.</p>
      ...
    </div>
  </body>
</tt>
```

See section Selecting Subtitle Tracks for how to reference specific subtitle languages from the SMIL file.

5.1.2 MPEG-DASH On-Demand Format Details

There are many types and flavors of MPEG-DASH VoD assets. The Recorder currently has support for the DASH-IF on-demand profile including side-loaded subtitle files. (See [6]).

In the example below, there is one subtitle track provided as a TTML file and another provided as a WebVTT file. Normally, one input format is used for all subtitles.

Example:

```
<?xml version="1.0"?>
<MPD xmlns="urn:mpeg:dash:schema:mpd:2011" minBufferTime="PT1.500S" type="
   ↪ static"
     mediaPresentationDuration="PT0H0M30.067S" maxSegmentDuration="PT0H0M4
        ↪ .000S"
     profiles="urn:mpeg:dash:profile:isoff-on-demand:2011">
 <ProgramInformation moreInformationURL="http://gpac.sourceforge.net">
  <Title>test30s</Title>
 </ProgramInformation>
```

```
 <Period duration="PT0H0M30.067S">
   <AdaptationSet contentType="video" segmentAlignment="true" maxWidth="960"
      ↪ maxHeight="540"
                  maxFrameRate="30" par="16:9" lang="und" subsegmentAlignment
                     ↪ ="true"
                  subsegmentStartsWithSAP="1">
   <Role schemeIdUri="urn:mpeg:dash:role:2011" value="main"/>
   <Representation id="1" mimeType="video/mp4" codecs="avc1.640029" width
      ↪ ="640"
                   height="360" frameRate="30" sar="1:1" startWithSAP="1"
                   bandwidth="304836">
    <BaseURL>video300kbps.mp4</BaseURL>
    <SegmentBase indexRangeExact="true" indexRange="915-1042">
      <Initialization range="0-914"/>
    </SegmentBase>
   </Representation>
   <Representation id="3" mimeType="video/mp4" codecs="avc1.640029" width
      ↪ ="960"
                   height="540" frameRate="30" sar="1:1" startWithSAP="1"
                   bandwidth="603350">
    <BaseURL>video600kbps.mp4</BaseURL>
    <SegmentBase indexRangeExact="true" indexRange="915-1042">
      <Initialization range="0-914"/>
    </SegmentBase>
   </Representation>
  </AdaptationSet>
  <AdaptationSet contentType="audio" segmentAlignment="true" lang="eng"
     ↪ subsegmentAlignment="true"
                 subsegmentStartsWithSAP="1">
   <Role schemeIdUri="urn:mpeg:dash:role:2011" value="main"/>
   <Representation id="2" mimeType="audio/mp4" codecs="mp4a.40.2"
      ↪ audioSamplingRate="48000"
                   startWithSAP="1" bandwidth="14764">
    <AudioChannelConfiguration
        schemeIdUri="urn:mpeg:dash:23003:3:audio_channel_configuration:2011"
           ↪ value="2"/>
    <BaseURL>audio.mp4</BaseURL>
    <SegmentBase indexRangeExact="true" indexRange="836-963">
      <Initialization range="0-835"/>
    </SegmentBase>
   </Representation>
  </AdaptationSet>
  <AdaptationSet contentType="text" mimeType="application/ttml+xml" lang="swe
     ↪ ">
    <Role schemeIdUri="urn:mpeg:dash:role:2011" value="subtitle"/>
    <Representation id="sub_swe" bandwidth="2000">
      <BaseURL>sub_swe.ttml</BaseURL>
    </Representation>
  </AdaptationSet>
  <AdaptationSet contentType="text" mimeType="text/vtt" lang="nor">
    <Role schemeIdUri="urn:mpeg:dash:role:2011" value="subtitle"/>
    <Representation id="sub_nor" bandwidth="2000">
      <BaseURL>sub_nor.vtt</BaseURL>
    </Representation>
  </AdaptationSet>
 </Period>
</MPD>
```

5.1.3 SMIL Format Details

SMIL files come in many types and flavors. The Recorder currently supports segmenting of MP4 VoD assets which are described using a subset of the Wowza SMIL format.

5.1.3.1 Simple SMIL File

In its most basic form, the SMIL file lists a number of media source paths and their individual bitrates.

```
<?xml version="1.0"?>
<smil>
  <head>
</head>
  <body>
    <switch>
      <video src="high.mp4" system-bitrate="2000000"/>
      <video src="medium.mp4" system-bitrate="1000000"/>
      <video src="low.mp4" system-bitrate="500000"/>
    </switch>
  </body>
</smil>
```

Since there is no information describing what tracks these files contain, or which of those tracks to include in the output, the Recorder will process every track that is found in each source file. Every track will yield a separate variant in the ABR output.

5.1.3.2 SMIL Files with Specific Tracks

5.1.3.2.1 Selecting Audio and Video Tracks

In order to extract a single media type from a source file, and also specify the bitrate and language for that variant, the `audioOnly` and `videoOnly` attributes are used. The example below creates an output with two variants at high bitrate and two variants at low bitrate:

```
<?xml version="1.0"?>
<smil>
  <head/>
  <body>
    <switch>
      <video src="high.mp4" system-bitrate="2000000">
        <param name="videoOnly" value="TRUE" valuetype="data"/>
      </video>
      <video src="high.mp4" audio-bitrate="128000">
        <param name="audioOnly" value="TRUE" valuetype="data"/>
      </video>
      <video src="low.mp4" system-bitrate="500000">
        <param name="videoOnly" value="TRUE" valuetype="data"/>
      </video>
      <video src="low.mp4" audio-bitrate="64000">
        <param name="audioOnly" value="TRUE" valuetype="data"/>
      </video>
    </switch>
  </body>
</smil>
```

If a source file contains more than one track of the same media type, e.g. two audio tracks, then an index is needed to uniquely identify each track that should be used. The track index is included in the `src` tag as a query string parameter:

```
<?xml version="1.0"?>
<smil>
  <head/>
```

```
  <body>
    <switch>
      <video src="mp4:movie.mp4?audioindex=0" audio-bitrate="128000" system-
          ↪ language="Mandarin">
        <param name="audioOnly" value="TRUE" valuetype="data"/>
      </video>
      <video src="mp4:movie.mp4?audioindex=1" audio-bitrate="128000" system-
          ↪ language="Korean">
        <param name="audioOnly" value="TRUE" valuetype="data"/>
      </video>
      <video src="movie.mp4" system-bitrate="2000000">
        <param name="videoOnly" value="TRUE" valuetype="data"/>
      </video>
    </switch>
  </body>
</smil>
```

Tracks are selected using the `audioindex` or `videoindex` query parameter, depending on the type of the track. The track indices are enumerated from 0 per media type in the order they appear in the MP4 file, e.g. the first audio track and the first video track both have the index 0.

Note that the asset filename should be prefixed with `mp4:` when a query parameter is used.

5.1.3.2.2 Selecting Subtitle Tracks

Subtitle tracks are added to the SMIL file by using the `textstream` tag which should reference a TTML file as its `src`. The language contained within each TTML file is specified as an ISO 639 code in the `system-language` attribute. If there are multiple languages available in a TTML file, a comma-separated list of language codes is used to specify which languages to include:

```
<textstream src="captions.ttml" system-language="ger,ita">
</textstream>
```

5.2 VoD in Convoy

5.2.1 Enabling VoD Repackaging in Configuration

The following parameters must be set in `convoy.conf` to repackage VoD uploads to to ESF:

```
ccmi.webtv_upload.storages = { storage1 }
ccmi.webtv_upload.repackage = true
```

`ccmi.webtv_upload.storages` specifies the storage IDs where the VoD content will be stored and `ccmi.webtv_upload.repackage` enables repackaging into ESF.

The upload and repackaging will take place on nodes with the `upload-agent` role (which defaults to the management nodes unless it has been explicitly defined), and log output will end up in journald. It can be read with:

```
    journalctl -u convoy-upload
```

5.2.2 VoD Upload Examples

5.2.2.1 MPEG-DASH On-Demand Source

An MPEG-DASH VoD file with the `.mpd` extension is required as location when uploading and repackaging MPEG-DASH VoD content into ESF.

Example:

HTTP

```
content upload -l http://mynas/asset/manifest.mpd ASSET_ID -t 'ASSET TITLE' -
    ↪ S HTTP
```

FTP

```
content upload -l ftp://user:pwd@mynas/asset/manifest.mpd ASSET_ID -t '
    ↪ ASSET_TITLE' -S HTTP
```

File

```
content upload -l file:///mynas/asset/manifest.mpd ASSET_ID -t 'ASSET TITLE'
    ↪ -S HTTP
```

5.2.2.2 MP4 Source

An MP4 SMIL file with the `.smil` extension is required as location when uploading and repackaging MP4 VoD content into ESF.

Example:

HTTP

```
content upload -l http://mynas/asset/asset.smil ASSET_ID -t 'ASSET TITLE' -S
    ↪ HTTP
```

FTP

```
content upload -l ftp://user:pwd@mynas/asset/asset.smil ASSET_ID -t '
    ↪ ASSET_TITLE' -S HTTP
```

File

```
content upload -l file:///mynas/asset/asset.smil ASSET_ID -t 'ASSET TITLE' -S
    ↪  HTTP
```

5.2.2.3 HLS Source

The master playlist file is specified as location when uploading a HLS VoD content.

Example:

HTTP

```
content upload -l http://mynas/asset/index.m3u8 ASSET_ID -t 'ASSET TITLE' -S
    ↪ HTTP
```

FTP

```
content upload -l ftp://user:pwd@mynas/asset/index.m3u8 ASSET_ID -t 'ASSET
    ↪ TITLE' -S HTTP
```

File

```
content upload -l file:///mynas/asset/index.m3u8 ASSET_ID -t 'ASSET TITLE' -S
    ↪  HTTP
```

6 Use Case: VOD Ingest using AWS S3 object storage

ESB3005 VoD Ingest on its own only supports ingest to a mounted file storage or FTP. In order to perform VoD ingest to S3 object storage, e.g. to an AWS S3 bucket, the ESB3005 ISO also bundle and installs the `s3fs-fuse` package. s3fs allow fuse mounts of S3 buckets, to allow it to be be treated as mounted storage.

To be able to use fuse mounts that may be mounted and accessible by any user, make sure that the following line exists in /etc/fuse.conf

```
user_allow_other
```

Mount an AWS S3 bucket:

```
mkdir /path/to/mountpoint
echo "<mybucket>:<myAccessKeyId>:<mySecretAccessKey>" >> $HOME/.passwd-s3fs
chmod 600 $HOME/.passwd-s3fs
s3fs <mybucket> /path/to/mountpoint -o passwd_file=$HOME/.passwd-s3fs -o
    ↪ umask=133 -o allow_other
```

Or mount as root via fstab:

```
mkdir /path/to/mountpoint
echo "<mybucket>:<myAccessKeyId>:<mySecretAccessKey>" >> /etc/passwd-s3fs
chmod 600 /etc/passwd-s3fs
echo "s3fs#<mybucket> /path/to/mountpoint fuse _netdev,allow_other 0 0" >> /
    ↪ etc/fstab
```

For further details about s3fs and additional options, see:

```
man s3fs
```

When s3fs is successfully mounted it is possible to use it via the mount point as any mounted file system. Please note that S3 is a HTTP based interface, and s3fs is not 100% POSIX compliant and some limitations apply on what you can and should do. The `ew-recorder` command is s3fs compliant.

Perform VoD ingest:

```
ew-recorder --input /path/to/source/content.smil --output /path/to/mountpoint
    ↪ /content [--content-id content] [--log-level info]
```

See [9] for further details about VoD ingest using ESB3005.

6.1 s3fs local storage

The use of a local cache directory for intermediate storage can be turned on or off in s3fs. Normally the cache should be disabled in a write-only use case for performing VOD Ingest/recordings. But even with the cache directory disabled, s3fs will utilize temporary local storage for any currently opened files towards the s3fs mount. This means that it is required to have enough local accessible disk space to fit the size of all concurrent recordings. And if the source files for VOD ingest are also on a S3 bucket, it is required to have twice the size of all concurrent recordings in available local disk space.

7 Use Case: VOD Ingest using Azure blob object storage

ESB3005 VoD Ingest on its own only supports ingest to a mounted file storage or FTP. In order to perform VoD ingest to blob object storage, e.g. to an Azure container, the ESB3005 ISO also bundles and installs the `blobfuse` package. Blobfuse allows fuse mounts of blob containers, to allow it to be be treated as mounted storage.

To be able to use fuse mounts that may be mounted and accessible by any user, make sure that the following line exists in `/etc/fuse.conf`

```
user_allow_other
```

7.1 Blobfuse cache directory

Blobfuse requires a cache directory for intermediate storage. The cache directory should preferably be on fast local storage or even RAM disk for best performance. The size of the storage where the cache directory resides, needs to be large enough to hold the size of all concurrent recordings performed on a single blobfuse mount.

7.2 Non-persistent user mount

Create the cache and mount directories:

```
mkdir -p /path/to/blobfusecachedir
mkdir -p /path/to/mountpoint
```

Create the blobfuse configuration file `$HOME/.blobfuse.cfg` containing:

```
accountName <account-name-here>
# Please provide either an account key or a SAS token, and delete the other
    ↪ line.
accountKey <account-name-here-delete-next-line>
sasToken <shared-access-token-here-delete-previous-line>
containerName <container-name-here>
```

Set access rights for the configuration file:

```
chmod 600 $HOME/.blobfuse.cfg
```

Mount the blob container using blobfuse:

```
blobfuse /path/to/mountpoint --config-file=$HOME/.blobfuse.cfg --tmp-path=/
    ↪ path/to/blobfusecachedir -o umask=133 -o allow_other
```

7.3 Persistent mount as root via fstab

Create mount script `/usr/bin/blobfusemount.sh` as root, with blobfuse specific options, containing:

```
#!/bin/bash
MOUNT_DIR=$1
/usr/bin/blobfuse $MOUNT_DIR --tmp-path=/path/to/blobfusecachedir --config-
    ↪ file=/etc/blobfuse.cfg -o allow_other
```

Make the mount script executable for root:

```
sudo chmod 744 /usr/bin/blobfusemount.sh
```

Create the cache and mount directories:

```
sudo mkdir -p /path/to/blobfusecachedir
sudo mkdir -p /path/to/mountpoint
```

Create the blobfuse configuration file `/etc/blobfuse.cfg` containing:

```
accountName <account-name-here>
# Please provide either an account key or a SAS token, and delete the other
    ↪ line.
accountKey <account-name-here-delete-next-line>
sasToken <shared-access-token-here-delete-previous-line>
containerName <container-name-here>
```

Set access rights for the configuration file:

```
sudo chmod 600 /etc/blobfuse.cfg
```

Add a line with a mount entry to `/etc/fstab`:

```
/usr/bin/blobfusemount.sh /path/to/mountpoint fuse _netdev,allow_other 0 0
```

Mount the container:

```
sudo mount /path/to/mountpoint
```

For further details about blobfuse and additional options, see https://github.com/Azure/azure-storage-fuse.

7.4 VoD ingest

When blobfuse is successfully mounted, it is possible to use it via the mount point as a mounted file system. Please note that the underlying API towards Azure blob storage is a HTTP based interface, and it is not 100% POSIX compliant and limitations apply on what you can and should do. The `ew-recorder` command is blobfuse compliant.

Perform VoD ingest:

```
ew-recorder --input /path/to/source/content.smil --output /path/to/mountpoint
    ↪ /content [--content-id content] [--log-level info]
```

See [9] for further details about VoD ingest using ESB3005.

8 Use Case: Recordings to AWS S3 object storage

The ESB3005 recorder on its own only supports recordings to a mounted file storage or FTP. In order to perform recordings from the Catch-up Buffer Manager service (ew-cbm) to S3 object storage, e.g. to an AWS S3 bucket, the ESB3005 ISO also bundles and installs the `s3fs-fuse` package. s3fs allow fuse mounts of S3 buckets, to allow it to be be treated as mounted storage.

Install ESB3005 on the same host as ESB3003 as described in [9].

See [VoD Ingest for AWS S3] for information on how to set up s3fs.

AWS s3 recording may then be performed by:

```
ew-recorder --cbm --host 127.0.0.1 --start-epoch <start time> --stop-epoch <
    ↪ stop time> --channel <live ingest channel> --output /path/to/
    ↪ s3fsmountpoint/content-name [--content-id <content id>] [--log-level <
    ↪ level>]
```

See [9] for further details about how to perform live recordings using ESB3005.

9 Use Case: Recordings to Azure blob storage

The ESB3005 recorder on its own only supports recordings to a mounted file storage or FTP. In order to perform recordings from the Catch-up Buffer Manager service (ew-cbm) to Azure blob storage, the ESB3005 ISO also bundles and installs the `blobfuse` package. blobfuse allow fuse mounts of Azure blob containers, to allow it to be be treated as mounted storage.

Install ESB3005 on the same host as ESB3003 as described in [9].

See [VoD Ingest for Azure blob] for information on how to set up blobfuse.

Azure blob storage recording may then be performed by:

```
ew-recorder --cbm --host 127.0.0.1 --start-epoch <start time> --stop-epoch <
    ↪ stop time> --channel <live ingest channel> --output /path/to/
    ↪ blobfusemountpoint/content-name [--content-id <content id>] [--log-
    ↪ level <level>]
```

See [9] for further details about how to perform live recordings using ESB3005.

10 Use Case: Low Latency Live Streaming

Refer to [12] for how to setup live channels. Currently, SW Origin supports Low Latency for DASH [13] and HLS [14].

10.1 End-to-End Latency

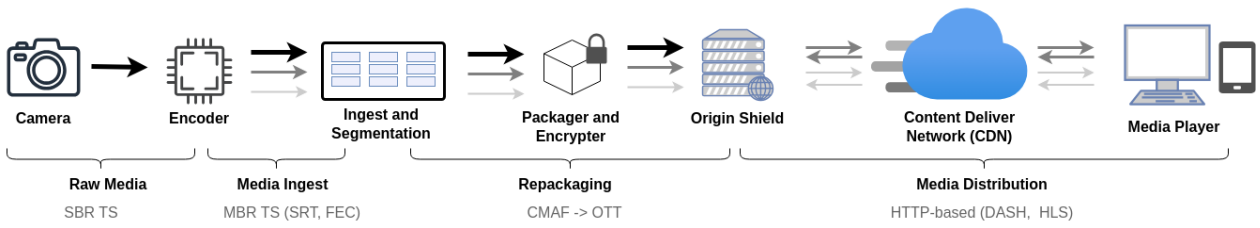


Figure 1: A typical media streaming architecture

Deploying a low latency live (LLL) streaming service requires a diligent examination and streamlining of every component in the media delivery pipeline. To this end, it is imperative to understand all the components and functionality that contribute to the E2E latency. As shown in above figure, the architecture consists of three main stages: media preparation, media ingest, and media distribution.

- **Media Ingest:** This stage involves acquiring multiple bitrates content from a source (encoder) and outputing segmented stream.
- **Packaging:** Delivers segmented output in OTT format. Often with DRM protection, such as Widevine.
- **Media Distribution:** This stage involves delivering content to consumers. Using HTTP for distribution, CDNs can shield the origin server and scale distribution to millions of media players on various platforms. CDN distribution latency is a key E2E latency component, influenced by CDN server locations and configuration. Another crucial factor, for the balance between latency and stability, is the player configuration of buffer size, rate adaptation, playhead positioning, license fetching, decryption, decoding, and rendering with minimal buffering under low-latency constraints.

10.2 Prerequisites

The table below summarizes the software versions used for this use case description.

System Component	Release version
ESB3003 SW Live Ingest	1.42.1
ESB3002 SW Repackager	1.52.1

10.3 Configuration

10.3.1 SW Live Ingest

To enable low latency for a channel, configure its corresponding segmentation template.

```
confcli services.liveIngest.segmentationTemplates.<template name>.lowLatency
```

For example, with segment duration of 4 seconds and the chunk duration to 500ms, each segment is divided into 8 chunks. As required from Low Latency HLS, keeping 3 live-edge segments, with 3 marginal segments. This ensures around 48 (8x6) latest chunks are available.

Channel `configId` has to be increased to apply the change. Let the channel run for a few seconds, and verify the changes by:

- Checking the live buffer: chunk_times.json for each reconfigured channel, and a `chunks/` folder contains chunks for each track.

```
/mnt/ramdisk/<channel>/
|-- cfg_0
|   |-- audio
|   |   |-- ...
|   |   |-- 432241259.cmfa
|   |   |-- chunks
|   |   |   |-- 432241259.0.cmfa
|   |   |   |-- ...
|   |   |-- init.cmfa
|   |-- ...
|-- chunk_times.json
|-- content_info.json
|-- manifest.mpd
|-- segment_times.json
```

- `chunk_times.json` can be requested:

```
curl "<live ingest ip>:8090/<channel>/chunk_times.json?segmentNum=0&
   ↪ chunkIndex=0"
```

NOTE: To support longer segments (e.g. 8 seconds or more), it is necessary to increase the `proxy_read_timeout` under `chunk_times` location scope of the CBM Accelerator configuration (`/etc/ew-cbm/ew-cbm-accelerator.conf`) to at least the segment duration.

10.3.2 SW Repackager

The Low Latency DASH/HLS stream is automatically output by the Repackager when the upstream SW Live Ingest activates low latency.

NOTE: To serve chunks to clients, increase the `readTimeoutMs` of `services.repackaging.locations.ingestServers.<server name>`. It should be at least the max segment duration necessary for HTTP chunked delivery. I.e at ingest servers, there are two in-used low latency segmentation templates with `exactAverageSegmentDuration`: 4 sec and 8 sec, then ingest servers `readTimeoutMs` should be at least 8000.

10.3.2.1 Low Latency DASH

In MPEG-DASH, segment templates are crucial in defining how media segments are referenced and accessed. Segment templates provide a flexible way to construct URLs for media segments and initialisation segments, using either numbering or time-based indexing. The segment template `SegmentNumber` is recommended over `SegmentTimeline`.

To further tune the manifest generation process change the presentation parameter:

- **dash.targetLatency** needs great care tunes to ensure smooth playback regardless of network fluctuations. However, the actual value in the MPD might be larger than the configured value because of the ingest stage at SW Live Ingest.

- **dash.suggestedPresentationDelay** is often skipped by players, with some exceptions like PRESTOplay that prefers this over target latency in low-latency mode.
- **dash.minimumUpdatePeriod** regulates how often clients check for manifest updates, optimizing network usage. If the segment template is **SegmentTimeline**, it is suggested to set **minimumUpdatePeriod** to a lower value.

10.3.2.2 Low Latency HLS

SW Repackager has limited support for output formats of Low Latency HLS. The presentation parameters `hls.segmentFormat` must be `fMP4`. Other values drop low latency tags out of playlist, regardless upstream activates low latency or not.

10.4 Validation

It depends on platform abilities to support certain codecs in encrypted LLL stream. And not all players enable or support LLL stream by default.

It is important to check the average buffer length and actual latency. A smooth playback experience requires a stable buffer length. The actual latency is subject to many other factors in the whole pipeline but the player should be able to adapt to the target latency specified in the manifest.

Some players that support LLL stream:

- For DASH: DASH JS, ExoPlayer and PRESTOplay.
- For HLS: Safari or HLS JS.

10.5 Challenges to Smooth Playback

Along the LLL streaming pipeline, several challenges may arise, including early requests causing HTTP 404 errors, late or missing segments disrupting playback, buffer underflows leading to rebuffering, bandwidth fluctuations impacting video quality, manifest errors causing playback interruptions, and CDN issues affecting content delivery. To address such problems:

- **Early Requests**: Request throttling mechanisms on the client side are needed to ensure that requests are made closer to the segment's availability time. Specifically, the Repackager returns 404 errors for early requests so the media player needs to implement retry logic and manage pending requests effectively.
- **Late/Missing Segments**: Similarly, the player needs to support retrying requests for missing segments, seamlessly switching to alternative representations, or skipping problematic segments while maintaining uninterrupted playback.
- **Buffer Underflow**: Buffer underflows happen when the playback buffer is depleted before the next segment arrives, causing interruptions. To address this, optimize buffer management by configuring media player settings to balance enough buffered media to prevent underflows while maintaining low latency. This involves adjusting buffer sizes and initial playhead positioning to ensure the buffer level stays above the safe threshold.
- **Bandwidth Fluctuations**: Maintaining a smooth LLL streaming service amidst variable bitrate requires delivering high-quality content with minimal rebuffering. The limited encoding time in LLL streaming necessitates efficient low-latency encoders and a carefully chosen bitrate ladder. To mitigate rebuffering caused by transient network conditions, buffering more media can help, though it may increase latency. Thus, streaming clients should use adaptive playback speed controllers to balance latency management and reduce prebuffering.
- **CDN Issues**: CDNs should implement caching strategies that minimize the caching of 404 errors to prevent cascading failures. Monitoring CDN performance and promptly addressing misconfigurations or caching issues can help maintain reliable media delivery. In case of misconfigurations or CDN failures, some failover mechanisms can be useful to ensure fault tolerance and reliable content delivery.
- **Codec support**: codec profile can affect latency in LLL stream, such as AVC High Profile, which introduces more B-frames, increase latency due to wait for data to decode.