agiletv

Live and Catchup with TV Content Capture and TV Repackaging

Document no	EDGS-184
Version	A5
Created on	2025-06-12

CONFIDENTIAL
©Copyright AgileTV, 2025

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 adaptive bitrate (ABR) streaming from live multicast or unicast ATS sources, using an esb3002 Software Repackager node in front of an esb3003 Software Live Ingest node.

1.3 References

[1] EDGS-187 - Software Live Ingest User Guide

[2] EDGS-176 - SW Repackager User Guide

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

[4] OpenCable Adaptive Transport Stream Specification OC-SP-ATS-I01-140214

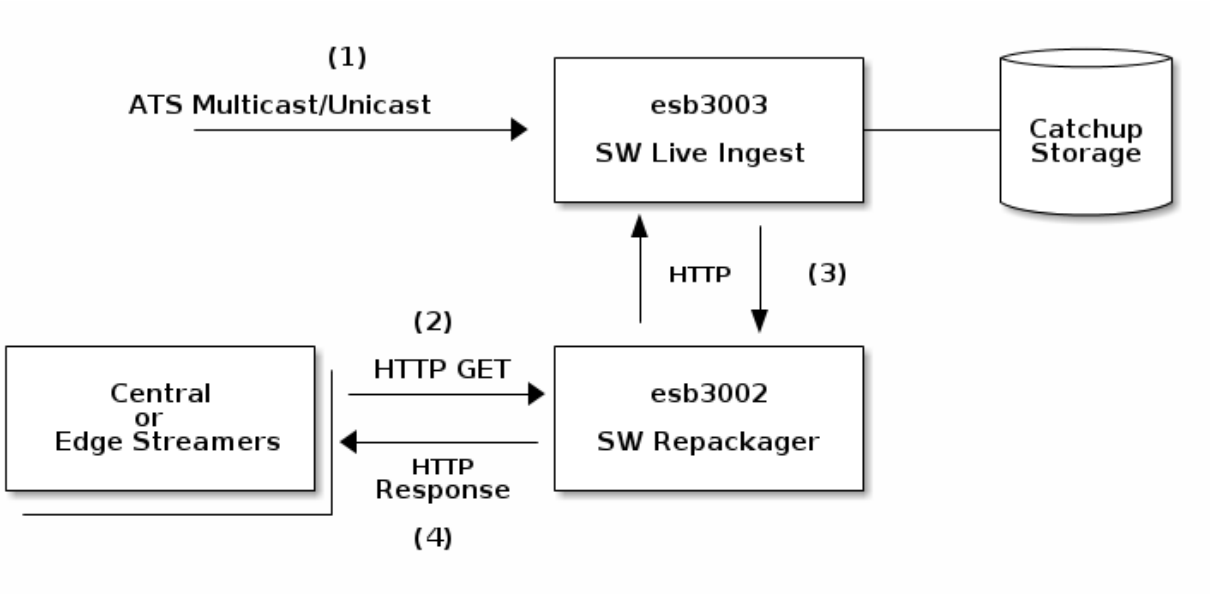
1.4 History

Version	Date	Changes
A1	2017-12-01	Initial version
A2	2018-08-28	Update for subtitles support
A3	2018-09-21	Input stream clarifications
A4	2022-02-22	Drop unmanaged VOD support
A5	2022-07-07	Update instruction to add channel content

2 Software Origin Live System

2.1 Overview

The image below shows an overview of a system where a Software Repackager, that is placed in front of a Software Live Ingest node, streams repackaged data to a group of clients. This document will describe how to configure and verify the functionality of this setup.



The basic data flow in this setup is as follows:

1. For each variant in a channel, a continuous ATS input stream enters the Software Live Ingest node. Here the variant streams are synchronized and media segments are created in the intermediate CMAF-based storage format ESF.
2. A client request for video manifest or segment is forwarded from an Edge or Central streamer to the Repackager node. This may be for either HLS, MPEG-DASH or Microsoft Smooth Streaming (MSS) format.
3. The Repackager retrieves the data necessary for serving the client from the Live Ingest node.
4. The media data is repackaged to the requested format and is delivered to the streamer for delivery and caching.

2.1.1 Segmentation

The segmentation algorithm is configured with video frame rate and exact (average) segment duration. The segmentation parameters in the SW Live Ingest segmenter must be aligned with the frame rate and GoP/segment duration configured in the encoder.

This results in a regular output segment stream with no media timeline drift, and timestamp alignment between channels with the same configuration. All video and subtitle segments will have the same duration. Audio segments may have slightly varying segment duration but with the same exact average duration as the video segments. An exception is segment boundaries moved due to ad insertion.

The media segments are stored on a RAM disk as CMAF segments by the segmenter from which the Catchup Buffer Manager fetches them and stores them as 1-minute CMAF tracks on disk or NAS. Accompanying metadata for fast retrieval is also stored. The combination of CMAF media data and metadata is called ESF format.

The publishing of segments is synchronized between the various tracks, so there is no misalignment that can result in player startup problems. However, this also means that varying arrival time of data or track will result in varying publish time. For the best possible system latency, all input media tracks should be aligned in the TS stream and audio, subtitles and video should arrive at roughly the same time. For further details regarding segment publish times and how they can be adjusted, refer to the Software Live Ingest User Guide [1].

2.1.2 Output formats

We support repackaging to three output formats HLS, DASH, and MSS from the SW Repackager ESB-3002. See the Software Repackager User Guide [2] for more details. Since we have changed DASH manifest format, it is described in some detail here.

2.1.2.1 DASH output

With the regular segment production, we now generate DASH manifests which use $Number$ with SegmentTemplate instead of SegmentTimeline.

This has many advantages and some drawbacks

Pros

- The manifest is very short, even for long timeShiftBuffer depth.
- The client does not need to fetch the manifest frequently
- All clients know that the clock is used to fetch the right segment. DASH provides UTCTiming for that.
- This is the most used format for live, and best tested
- Good for CDN ingestion

Cons

- Client and server clocks need to be synchronized
- emsg-messages need to be used to for a quick manifest update
- There will be more HTTP requests resulting in 404/412 responses due to early requests, or late segment generation.

2.1.3 Configuration Changes and configId

Since live channels are typically long-lived, it is possible that they need to be reconfigured at some time. This could, for example be, changes of the encoder settings, the set of video bitrate variants, or the segment durations. Such changes often result in output streams which are different compared to the state before the change. For example, removing a video bitrate from a channel means that the HLS master playlist becomes invalid for that channel. It is therefore, in general, not possible to have streaming sessions that cross such a configuration change border. Still, it is important that the catchup-buffer for the time interval before the change is still available for streaming.

To allow for configuration changes in either the encoded streams or the chosen channel parameters, the parameter `configId` has been introduced. This must be configured for each channel, and must be changed if the channel configuration or its media is changed in such a way that the output is no longer compatible with its previous form. When changing the configuration for a channel or the encoder settings for the channel, one should be aware if the output form will chang and if that is desired. Therefore, this value must be changed manually. If not changed, but the output is not compatible with the previous configuration there will be an alert and corresponding logging messages. One can then choose to revert the setting, or change the configId to allow for this non-compatible change.

See the user guide [1] for more information about initial and later configuration of the Software Live Ingest node.

2.2 Installation

Please refer to the user guides for Software Live Ingest [1] and Software Repackager [2] for information on how to install these products.

2.3 Configuration

The following sections will describe how to set up and verify the Software Origin system described above.

In order to complete the setup, the following is required:

- One RedHat 7/8 node with esb3002 installed
- One RedHat 7/8 node with esb3003 installed
- OpenCable-compliant ATS multicast/unicast in the network [4]
- System clock among the nodes must be synchronized

Please refer to the user guides for Software Live Ingest [1] and Software Repackager [2] for details about the configuration parameters that are used in this section.

2.3.1 Creating a RAM disk for Live Ingest

By default the Live Ingest service writes its output to `/mnt/ramdisk`. As the name implies this location should be a RAM disk. The size required for the RAM disk size depends on the total ingest bandwidth and the configured circular buffer length. For example, ingesting fifty channels where each channel uses a bandwidth of 12Mbps and the `circularBufferS` configured to 60 seconds would require a RAM disk size of minimum 4500MB.

```
RamDiskSizeInMB = (12 * 50 * 60) / 8 = 4500
```

The following commands can be used to create a 1 GB RAM disk:

```
[root@esb3003 ~]# mkdir -p /mnt/ramdisk
[root@esb3003 ~]# mount -t tmpfs -o size=1G tmpfs /mnt/ramdisk
```

2.3.2 Adding a Software Live Ingest channel

After a fresh install the live ingest configuration will be empty. Check the configuration by logging on to the esb3003 server and issue the command `confcli services.liveIngest`:

```
[root@esb3003 ~]# confcli services.liveIngest
{
    "liveIngest": {
        "channelTemplates": [],
        "channels": [],
        "logging": {
            "general": "INFO",
            "source": "INFO",
            "track": "INFO"
        },
        "segmentationTemplates": []
    }
}
```

We will start by adding a new live channel on the Software Live Ingest node. Since this is the first channel, a SegmentationTemplate and a ChannelTemplate need to be defined first.

2.3.2.1 Segmentation template

The SegmentationTemplate should be set up to match the following properties of the input stream:

- The framerate of the video should match
- The `exactAverageSegmentDuration` should be a multiple of the key frame distance
- The `segmentationTrigger` should match the segmentation marker type that is used in the ATS input

In this example, a SegmentationTemplate that produces four-second segments will be created. The video framerate is 25 fps and the segmentation trigger is RAI.

Type `confcli -w services.liveIngest.segmentationTemplates` to enter the confcli wizard:

```
[root@esb3003 ~]# confcli -w services.liveIngest.segmentationTemplates
Running wizard for resource 'Segmentation Templates'
<A list of segmentation templates>

Hint: Hitting return will set a value to its default.
Enter '?' to receive the help string

Segmentation templates <A list of segmentation templates>: [
  Segment <Configuration for a single segmentation template>: {
    Video Frame Rate (default: 25): 25
    Segmentation Template Name (default: ): default
```

```
    Embed Ingest Timestamp in Video (default: True): True
    Max Skew for Media Components (default: 2000): 2000
    Exact Average Segment Duration (default: 4000): 4000
    Segmentation Timescale (default: 1000): 1000
    Circular Buffer Length (default: 60): 60
    Reference Time Vs Epoch (default: 0): 0
    Segmentation Trigger (default: rai): rai
  Add another 'segmentationTemplate' element to array 'segmentationTemplates
     ↪ '? [y/N]: n
Generated config:
{
  "segmentationTemplates": [
    {
      "videoFrameRate": "25",
      "name": "default",
      "embedIngestTimestampInVideo": true,
      "maxSkewForMediaComponentMs": 2000,
      "exactAverageSegmentDuration": 4000,
      "segmentationTimescale": 1000,
      "circularBufferS": 60,
      "referenceTimeVsEpochS": 0,
      "segmentationTrigger": "rai"
    }
  ]
}
Merge and apply the config? [y/n]: y
```

Press y and enter to store the SegmentationTemplate.

2.3.2.2 Channel template

A channel template defines a set of media properties, such as bitrate, language, and the PID, that are identical within a set of channels.

The properties listed in the ChannelTemplate must match those of the input stream. In our example, the ChannelTemplate matches input streams with 4 Mbps video on PID 33 and 128 kbps Chinese audio on PID 34.

Leave the language tag empty for video tracks.

The Teletext Filter below only applies to subtitle tracks. It does not need to exist for non-subtitle tracks, but has to be added when using the wizard. Just accept the default values.

Type `confcli -w services.liveIngest.channelTemplates` to enter the confcli wizard:

```
[root@esb3003 ~]# confcli -w services.liveIngest.channelTemplates
Running wizard for resource 'Channel Templates'
<A list of channel templates>

Hint: Hitting return will set a value to its default.
Enter '?' to receive the help string

Channel templates <A list of channel templates>: [
  Channel template <Configuration shared by all channels that use this
     ↪ template>: {
    Tracks <A list of tracks>: [
      Track <Configuration for a single track>: {
        Language (default: ):
        Bitrate (default: 0): 4000000
        Track Name (default: ): video1
        Log Level (default: INHERIT):
        PID (default: -1): 33
        Track Media Type (default: video): video
        Teletext Filter List <A list of at most one teletext filter>: [
```

```
          Teletext Filter <Specifies which teletext input from the pid to use
             ↪  ...: {
            Magazine (default: -1):
            Type (default: ):
            Page (default: ):
          Add another 'teletextFilter' element to array 'teletextFilter'? [y/
             ↪ N]:
      Add another 'track' element to array 'tracks'? [y/N]: y
      Track <Configuration for a single track>: {
        Language (default: ): chi
        Bitrate (default: 0): 128000
        Track Name (default: ): audio1
        Log Level (default: INHERIT):
        PID (default: -1): 34
        Track Media Type (default: video): audio
        Teletext Filter List <A list of at most one teletext filter>: [
          Teletext Filter <Specifies which teletext input from the pid to use
             ↪  ...: {
            Magazine (default: -1):
            Type (default: ):
            Page (default: ):
          Add another 'teletextFilter' element to array 'teletextFilter'? [y/
             ↪ N]:
      Add another 'track' element to array 'tracks'? [y/N]: n
    Channel Template name (default: ): default
  Add another 'channelTemplate' element to array 'channelTemplates'? [y/N]: n
Generated config:
{
  "channelTemplates": [
    {
      "tracks": [
        {
          "lang": "",
          "bitrateBps": 4000000,
          "name": "video1",
          "logLevel": "INHERIT",
          "pid": 33,
          "mediaType": "video",
          "teletextFilter": [
            {
              "magazine": -1,
              "type": "",
              "page": ""
            }
          ]
        },
        {
          "lang": "chi",
          "bitrateBps": 128000,
          "name": "audio1",
          "logLevel": "INHERIT",
          "pid": 34,
          "mediaType": "audio",
          "teletextFilter": [
            {
              "magazine": -1,
              "type": "",
              "page": ""
            }
          ]
        }
      ],
```

```
      "name": "default"
    }
  ]
}
Merge and apply the config? [y/n]:
```

Press y and enter to store the ChannelTemplate.

2.3.2.3 Catchup location

All channels need to have a location for storing media data for catchup requests. To support a long catchup buffer, of e.g. a couple of days, a large and reliable NAS is required. The example below will configure a catchup buffer named `catchup1` with a length of two hours available at the path `/media`.

Enter `confcli -w storage.catchup.locations` to add a new catchup location to the setup:

```
[root@esb3003 ~]# confcli -w storage.catchup.locations
Running wizard for resource 'Catchup Locations'
<A list of catchup locations.>

Hint: Hitting return will set a value to its default.
Enter '?' to receive the help string

Catchup Locations <A list of catchup locations.>: [
  location <Catchup location configurations.>: {
    Catchup Duration (default: 0): 7200
    Catchup Base Path (default: ): /media
    Catchup Location Name (default: ): catchup1
  Add another 'location' element to array 'locations'? [y/N]: n
Generated config:
{
  "locations": [
    {
      "duration": 7200,
      "basePath": "/media",
      "name": "catchup1"
    }
  ]
}
Merge and apply the config? [y/n]: y
```

2.3.2.4 Adding the channel

With the SegmentationTemplate, the ChannelTemplate and the CatchupLocation in place, we can create a channel that uses them. For this we need to know the multicast address of each of the ATS inputs to the channel, or the unicast destination address and associated port, matching the local address of a configured network interface on the Software Live Ingest node. In reality each channel will often receive several input streams, but to simplify this description we settle with specifying only one multicast address.

Enter `confcli -w services.liveIngest.channels` to add a channel to the list of live ingest channels:

```
[root@esb3003 ~]# confcli -w services.liveIngest.channels
Running wizard for resource 'Channels'
<A list of channels>

Hint: Hitting return will set a value to its default.
Enter '?' to receive the help string

Channels <A list of channels>: [
  Channel <Configuration for a single channel>: {
    Segmentation Template Name (default: ): default
```

```
    Logging <Logging configuration>: {
      Track Log Level (default: INHERIT):
      Source Log Level (default: INHERIT):
      General Log Level (default: INHERIT):
    Channel Name (default: ): channel1
    Channel Template Name (default: ): default
    Input Sources <A list of input sources>: [
      Input Source <Input source configuration>: {
        IgmpV3Source (default: ):
        Source (default: ): 224.9.8.7:9876
        Tracks <A list of tracks from the channel template>: [
          Track <Configuration for a single track>: {
            Log Level (default: INHERIT):
            Track Name (default: ): video1
          Add another 'track' element to array 'tracks'? [y/N]: y
          Track <Configuration for a single track>: {
            Log Level (default: INHERIT):
            Track Name (default: ): audio1
          Add another 'track' element to array 'tracks'? [y/N]: n
      Add another 'inputSource' element to array 'inputSources'? [y/N]: n
    Catchup Location (default: ): catchup1
    Config Id (default: 0): 1
  Add another 'channel' element to array 'channels'? [y/N]: n
Generated config:
{
  "channels": [
    {
      "segmentationTemplate": "default",
      "logging": {
        "track": "INHERIT",
        "source": "INHERIT",
        "general": "INHERIT"
      },
      "name": "channel1",
      "channelTemplate": "default",
      "inputSources": [
        {
          "igmpV3Source": "",
          "source": "224.9.8.7:9876",
          "tracks": [
            {
              "logLevel": "INHERIT",
              "name": "video1"
            },
            {
              "logLevel": "INHERIT",
              "name": "audio1"
            }
          ]
        }
      ],
      "catchupLocation": "catchup1",
      "configId": 1
    }
  ]
}
Merge and apply the config? [y/n]: y
```

2.3.2.5 Verifying the channel configuration

If the ATS input stream is received correctly the `ew-live-ingest` service will start segmenting the input and

write files to the live buffer located at `/mnt/ramdisk` in the following format:

```
/mnt/ramdisk/channel1/
|-- cfg_1
|   |-- audio1
|   |   |-- 377522482.cmfa
|   |   |-- 377522483.cmfa
|   |   |-- ...
|   |   |-- 377522498.cmfa
|   |   `-- init.cmfa
|   |-- content_info.json
|   |-- manifest.mpd
|   `-- video1
|       |-- 377522482.cmfv
|       |-- 377522483.cmfv
|       |-- ...
|       |-- 377522499.cmfv
|       `-- init.cmfv
|-- content_info.json
|-- manifest.mpd
`-- segment_times.json
```

The tools `ew-live-ingest-tool` and `ew-cb-media` can be used to inspect configured channels, see ew-cb-media and ew-live-ingest-tool for further information.

2.3.2.6 Troubleshooting

If no segments are created in `/mnt/ramdisk/` or if the message `No segment update for 20 sec` can be seen in `/var/log/edgeware/ew-cbm/ew-cbm.log`, then there might be a problem with receiving the ATS input stream.

If a multicast input source is used, make sure that there is a multicast route specified for the interface where multicast should be received. The last line of the following output is a multicast route:

```
[root@esb3003 ~]# ip route
default via 10.16.63.1 dev ens3
10.16.0.0/18 dev ens3  proto kernel  scope link  src 10.16.48.127
10.16.63.1 dev ens3  scope link
10.16.64.0/18 dev ens4  proto kernel  scope link  src 10.16.114.127
224.0.0.0/4 dev ens4  scope link
```

If there is no route for multicast traffic (224.x.y.z) one can be added with the `ip route` command:

```
ip route add 224.0.0.0/4 dev <interface name>
```

If there still are no segments created in `/mnt/ramdisk` there can be some compatibility issues with the content of the input stream. Some possible problems are:

- Wrong type of EBP markers
- Mismatch in the language tags of the stream and the ChannelTemplate
- Unsupported codec

To pinpoint the problem, view the contents of the log file for the channel, in this case `/var/log/edgeware/ew-live-ingest/live-ingest-channels/worker-channel1.log`.

To increase the verbosity of the live ingest log you can set the log level to DEBUG for the different elements in `services.liveIngest.logging`:

```
[root@esb3003 ~]# confcli services.liveIngest.logging.source DEBUG
[root@esb3003 ~]# confcli services.liveIngest.logging.general DEBUG
[root@esb3003 ~]# confcli services.liveIngest.logging.track DEBUG
```

Remember to lower the logging levels before starting ingest of multiple channels. The log levels can be reset to their default values with the following command:

```
[root@esb3003 ~]# confcli -d services.liveIngest.logging
```

The Catchup Buffer Manager service, `ew-cbm`, monitors all live channels. When new segments are created this can be seen in `/var/log/edgeware/ew-cbm/ew-cbm.log`:

```
2017-11-07 22:26:24,773 - cbm.livemonitor - INFO - LiveMonitor - Ch: channel1
    ↪  - Storing live segments 377522482 -> 377522495
2017-11-07 22:26:28,761 - cbm.livemonitor - INFO - LiveMonitor - Ch: channel1
    ↪  - Storing live segments 377522482 -> 377522496
...
```

The `ew-cbm` service provides an HTTP interface at port 8090 of the live ingest node. This is the location that the Software Repackager connects to to receive information about the available live channels.

If you cannot connect to port 8090 you need to open this port in the firewall:

```
[root@esb3003 ~]# iptables -I INPUT -p tcp -m tcp --dport 8090 -j ACCEPT
[root@esb3003 ~]# service iptables save
```

We can use `curl` to see the available channels. In our case `channel1` should be available. In this example 10.16.48.127 is the IP address of the live ingest node.

```
[user@host ~]# curl 10.16.48.127:8090/channels
[
  "channel1"
]
```

To see the start and stop time for the live buffer we can request the file `cb_info.json` within the channel:

```
[user@host ~]# curl 10.16.48.127:8090/channel1/cb_info.json
{
  "version": "0.1",
  "start_epoch_time": 1510089928,
  "end_epoch_time": 1510089988
}
```

2.3.3 Software Repackager

2.3.3.1 Add channel content

In order to stream content that is available at an upstream live ingest node, a `ingestServers` of the `channelGroups` content needs to be added to the Software Repackager.

Log on to the esb3002 node and enter the following command to add a new content named `live`:

```
[root@esb3002 ~]# confcli -w services.repackaging.locations.ingestServers.
Running wizard for resource 'Ingest Servers'
<Live Ingest servers from which to serve live, catchup and start-over content
    ↪ >

Hint: Hitting return will set a value to its default.
Enter '?' to receive the help string

Ingest Servers <Live Ingest servers from which to serve live, catchup and
    ↪ start-over content>: [
  ingestServer : {
```

```
    Name (default: ): srv-0
    URL (default: ): http://10.16.48.127:8090
    Connection Timeout (default: 500):
    Read Timeout (default: 1000):
    Send Timeout (default: 2000):
  Add another 'ingestServer' element to array 'ingestServers'? [y/N]: y
  ingestServer : {
    Name (default: ): srv-1
    URL (default: ): http://10.16.48.127:8090
    Connection Timeout (default: 500):
    Read Timeout (default: 1000):
    Send Timeout (default: 2000):
  Add another 'ingestServer' element to array 'ingestServers'? [y/N]:
Generated config:
{
  "ingestServers": [
    {
      "name": "srv-0",
      "url": "http://10.16.48.127:8090",
      "connectTimeoutMs": 500,
      "readTimeoutMs": 1000,
      "sendTimeoutMs": 2000
    },
    {
      "name": "srv-1",
      "url": "http://10.16.48.127:8090",
      "connectTimeoutMs": 500,
      "readTimeoutMs": 1000,
      "sendTimeoutMs": 2000
    }
  ]
}
Merge and apply the config? [y/n]: y

[root@esb3002 ~]# confcli -w services.repackaging.content.channelGroups
Running wizard for resource 'Channel Groups'
<Configuration of source locations for live, catchup and start-over served by
   ↪  circular buffer>

Hint: Hitting return will set a value to its default.
Enter '?' to receive the help string

Channel Groups <Configuration of source locations for live, catchup and start
   ↪ -over served by circular buffer>: [
  channelGroup : {
    Name (default: ): live
    Locations <List of Ingest servers>: [
      location (default: ): srv-0
      Add another 'location' element to array 'locations'? [y/N]: y
      location (default: ): srv-1
      Add another 'location' element to array 'locations'? [y/N]:
    Output profiles <List of allowed output profiles>: [
      outputProfile (default: default): default
      Add another 'outputProfile' element to array 'outputProfiles'? [y/N]:
  Add another 'channelGroup' element to array 'channelGroups'? [y/N]:
Generated config:
{
  "channelGroups": [
    {
      "name": "live",
      "locations": [
        "srv-0",
```

```
        "srv-1"
      ],
      "outputProfiles": [
        "default"
      ]
    }
  ]
}
Merge and apply the config? [y/n]: y
```

In this example 10.16.48.127 is the IP address for the live ingest node. The Software Repackager supports failover between sources, but here we specify the same node as both `Location` and `Backup Location`.

2.3.3.2 Test streaming

When requesting data from the Software Repackager, the CCMI URL format is used. In our example the content location, `__cl`, will be the channel source `live` prefixed with `cg:` signifying that this is a channel group. The content name, `__c`, will be `channel1`.

For more information about the CCMI URL format please see the CCMI Specification [3].

We will use the `default` output profile, `__op`. The `default` output profile is included in the configuration when installing the Software Repackager and does not use encryption.

We can now use `curl` to fetch the DASH live manifest for the live channel. In this example 10.16.47.213 is the IP address for the Software Repackager node:

```
curl 10.16.47.213/__cl/cg:live/__c/channel1/__op/default/__f/manifest.mpd
```

If the manifest can be received successfully you can try streaming the URL in a DASH player.

For more details about sources and output profiles, please see the Software Repackager User Guide [2].

2.3.3.3 Troubleshooting

If you cannot connect to the Software Repackager, make sure that port 80 is open in the firewall.

More details about repackaging requests can be found in the logs located in `/var/log/edgeware/ew-repackager` ↪ `/`.

The log level for the Software Repackager can be changed under `services.repackaging.logging`. For example:

```
[root@esb3002 ~]# confcli services.repackaging.logging.repackaging.level
   ↪ debug
```

2.4 Tools

2.4.1 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 [2] 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.

2.4.2 ew-cb-media

The `ew-cb-media` tool included in esb3003 can be used to list the catchup buffers. When run without parameters, the output looks like shown below.

```
Channel          Start time           End time             Duration
-------------------------------------------------------------------
channel1         Sep-21 12:00:00      Sep-21 15:44:24       3h 44m 24s
channel2         Sep-21 12:00:00      Sep-21 15:44:24       3h 44m 24s
...
```

2.4.3 ew-live-ingest-tool

The `ew-live-ingest-tool` tool included in esb3003 can be used to list the ingest status of all channels. When run with the `--list` parameter, the output looks like shown below.

```
Channel      State    Status   Sync  PID    Start Time                Restarts
-------------------------------------------------------------------------------
channel1     Enabled  Running  n/a   19411  2018-09-21 12:00:00 UTC
channel2     Enabled  Running  n/a   19411  2018-09-21 12:00:00 UTC
...
```

Please read the Monitoring section in [1] for further details on how to run `ew-live-ingest-tool`