



User Guide

AWS Elemental MediaTailor



AWS Elemental MediaTailor: User Guide

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon. All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.

Table of Contents

What is AWS Elemental MediaTailor?	1
Origin server requirements	1
MediaTailor concepts	2
HLS playlist types	2
DASH manifest types	4
Ad insertion concepts	9
Channel assembly concepts	11
How ad insertion works	12
Ad insertion event flow	13
Related services	16
Accessing MediaTailor	17
Pricing	17
Regions	17
Setting up	18
Sign up for an AWS account	18
Getting started with MediaTailor	19
Getting started with MediaTailor ad insertion	19
Prerequisites	20
Step 1: Access MediaTailor	20
Step 2: Prepare a stream	20
Step 3: Configure ADS request URL and query parameters	22
Step 4: Create a configuration	23
Step 5: Test the configuration	25
Step 6: Send the playback request to MediaTailor	26
(Optional) step 7: Monitor activity	27
Step 8: Clean up	28
Getting started with MediaTailor channel assembly	29
Prerequisites	29
Step 1: Create a source location	30
Step 2: Add VOD sources to your source location	31
Step 3: Create a channel	32
Step 4: Add programs to your channel's schedule	33
Step 5 (<i>optional</i>): Use MediaTailor to insert personalized ads into your stream	35
Step 6: Start your channel	36

Step 7: Test your channel	36
Step 8: Clean up	36
Inserting ads	38
Supported audio and video codecs	39
Understanding ad insertion behavior	39
Ad stitching behavior for VOD	39
Live ad stitching behavior	42
Server-guided ad insertion overview	45
Enable in the playback configuration	45
Create a server-guided session	45
Ad tracking with SGAI	47
Guided prefetch with manifest heartbeating	47
SGAI feature compatibility	48
SGAI live configuration	50
SGAI VOD configuration	53
Ad server integration requirements	57
VAST requirements	58
VPAID requirements	60
Configuration management	61
Creating a configuration	61
Viewing configuration details	71
Editing configuration settings	71
Deleting configurations	71
Integrating a content source	72
Input source requirements	72
Integrating an HLS source	73
Integrating an MPEG-DASH source	84
Securing origin interactions with SigV4	102
Integrating with Google's ad platforms	106
Server-side integration	107
Client-side integration	109
Optimizing ad fill rate	110
Customizing ad break behavior with ad break suppression	112
Configuring ad break suppression	112
Bumper ad insertion	126
Configuring bumpers	126

Pre-roll ad insertion	127
Slate ad insertion	128
Configuring the slate	129
How personalization threshold works	129
Slate configuration and VPAID	133
Ad transcoding	133
Dynamic transcoding	133
Custom transcode profiles	135
Prefetching ads	139
How prefetching works	140
Creating prefetch schedules	144
TPS-based traffic shaping	151
Deleting prefetch schedules	152
Preconditioned ads	152
Preconditioned ads requirements	153
Preconditioned ads workflow	156
Monetization Functions	157
How Functions fit into the MediaTailor workflow	157
Quick start guide	158
Lifecycle hooks	162
Function types and composition	170
Creating and managing	180
JSONata expression reference	183
Function examples	194
Troubleshooting and monitoring	198
Limits	207
Dynamic ad variables	210
Session variables	211
Player variables	225
Domain variables	227
Configuration aliases	228
Passing ADS parameters	236
Parameter routing	239
MediaPackage integration	241
Session behavior	243
Parameter reference	244

Parameter troubleshooting	246
Alias troubleshooting	250
MediaTailor manifest query parameters	253
Origin query parameters	254
MediaTailor character restrictions	255
MediaTailor HLS implicit sessions	256
MediaTailor DASH implicit sessions	258
MediaTailor explicit session initialization	260
MediaTailor protocol-specific behavior	261
MediaTailor CDN integration	264
Reporting ad tracking data	265
Server-side tracking	266
Client-side tracking	281
Overlay ads	358
Prerequisites for using overlay ads	359
Getting started	360
Logging and metrics	373
Billing for overlay ads in MediaTailor	375
Ad ID decoration	375
Session State	377
Manifests and ad metadata insertion	378
Ad Decision Server (ADS) interactions	402
Client-side tracking API	403
Creating linear assembled streams	405
Channel Assembly workflow types	405
VOD-to-live	406
Live-to-live	409
VOD + Live-to-live (Mixed)	412
Choosing the right workflow	415
Next steps	416
Working with source locations	417
Creating a source location	417
Configuring authentication for your source location	419
Working with VOD sources	428
Working with live sources	432
Using package configurations	437

Manifest caching	437
Working with channels	437
Create a channel	438
Using source groups with your channel's outputs	440
Delete a channel	441
Adding a program	441
Creating a program	442
Defining audience cohorts and alternate content	449
Generating audience-specific manifests	452
Insert ads and ad breaks	453
Set up ad insertion	453
SCTE-35 messages for ad breaks	454
Time-shifting a channel's playback	460
Using a CDN	462
CDN selection	463
Plan CDN integration	464
Estimate traffic requirements	466
Configure optimization strategies	467
Customize planning	468
Set up monitoring and scaling	470
Optimize costs	472
Test your implementation	473
Troubleshoot common issues	475
Set up CDN integration	477
CDN integration components and requirements	478
Prerequisites for CDN integration	20
CDN integration setup steps	479
Required headers for MediaTailor CDN integration	479
Set up CDN routing behaviors	481
Set up CDN mapping	485
Security best practices	487
Ad insertion with CDN	489
What you'll need	490
Before you begin	490
Benefits of CDN integration	491
Understand CDN architecture	492

Set up basic ad insertion	498
SSAI with channel assembly	501
Optimize CDN performance	508
Monitor CDN operations	511
Troubleshoot ad insertion with CDNs	516
Channel assembly with CDN	520
What you'll need	520
Before you begin	521
Benefits of CDN integration	522
Understand CDN architecture	523
Basic setup	525
Configure base URLs	526
Implement ad insertion	527
Configure time-shifted viewing	529
Monitor CDN operations	530
Complete optimization guide	531
MediaPackage CDN integration	531
Understanding the MediaPackage and CDN workflow	532
Prerequisites	532
Essential settings configuration	533
Integration verification	535
Next steps	536
Manifest filtering	537
CDN caching	542
CDN performance	546
CDN integration troubleshooting	551
CloudFront integration	556
Basic CloudFront setup	556
Performance optimization	563
Multi-Region resilience	566
Monitoring and troubleshooting	570
Third-party CDN setup	575
Prerequisites	576
Configure CDN caching rules	577
Verify your CDN configuration	579
Optimize CDN performance	580

Troubleshoot third-party CDN issues	582
CDN performance optimization	583
Caching optimization	584
Request routing optimization	588
Performance benchmarks	588
Advanced optimization	590
CDN monitoring	593
Essential CDN performance metrics	593
Set up monitoring tools	595
Implementation checklist	596
Related topics	597
CDN integration testing	597
Testing prerequisites	598
Systematic testing methodology	599
Pre-deployment checklist	603
Testing tools reference	605
Troubleshoot CDN integration	606
Quick diagnostic checklist	606
Integration testing	608
Manifest 404 errors	612
Manifest delivery issues	613
Segment delivery issues	614
Session management issues	616
Ad break timing issues	618
Performance and latency issues	620
Inconsistent device behavior	626
Troubleshooting preparation	627
Workflow-specific troubleshooting guides	629
CDN integration log analysis reference	631
CDN log interpretation	631
Error codes reference	634
Log analysis tools	635
MediaTailor logging configuration resources	636
CloudFormation automation	637
Why use AWS CloudFormation	638
Prepare for deployment	639

Deploy the template	641
Use the deployed resources	643
Test and validate your deployment	645
Troubleshoot deployment issues	646
Customize the template	648
Template reference	652
Production CloudFront configuration	657
Three-origin architecture	657
Cache behavior precedence and configuration	659
Get CDN integration support	665
When to escalate to Support	666
Gather information for support	666
Support resources	667
Support case best practices	668
Security	669
Data protection	670
Data encryption	671
Identity and Access Management	671
Audience	672
Authenticating with identities	672
Managing access using policies	673
How AWS Elemental MediaTailor works with IAM	675
Identity-based policy examples	680
Resource-based policy examples	683
AWS managed policies	683
Using service-linked roles	685
Troubleshooting identity and access	688
Compliance validation	691
Resilience	691
Infrastructure security	692
Cross-service confused deputy prevention	692
Logging and monitoring	693
CloudWatch Alarms	693
CloudTrail logs	693
AWS Trusted Advisor	694
Monitoring and tagging	695

Viewing logs	696
ADS logs	696
Manifest logs	754
Transcode logs	760
Using vended logs	763
Writing logs to CloudWatch Logs	767
Controlling the volume of ad insertion session logs	776
Filtering logs and events	778
Generating debug logs	781
Monitoring with CloudWatch metrics	786
AWS Elemental MediaTailor CloudWatch metrics	787
AWS Elemental MediaTailor CloudWatch dimensions	802
Using metrics to diagnose stale manifests	803
Recording API calls	806
AWS Elemental MediaTailor information in CloudTrail	807
Understanding AWS Elemental MediaTailor log file entries	808
Receiving Channel Assembly alerts	810
Troubleshooting manifest-related alerts	815
Viewing alerts	815
Handling alerts	816
Tagging resources	816
Supported resources	817
Tag restrictions	817
Managing tags	817
Workflow monitor	817
Components of workflow monitor	819
Supported services	820
Configuring workflow monitor	820
Using workflow monitor	839
Dual-stack endpoints	842
What are dual-stack endpoints?	842
Supported features	842
Dual-stack endpoint format	842
Using dual-stack endpoints	843
AWS Management Console	843
AWS Command Line Interface	843

AWS SDK	843
Considerations	844
Quotas	845
Quotas on ad insertion	845
Quotas on channel assembly	850
Troubleshooting	855
Event flow troubleshooting	855
Incomplete flows	855
Timing analysis	857
Common problems	859
Monitoring best practices	862
Additional resources	863
Playback troubleshooting	864
Client errors	865
Server errors	866
Examples	868
Beacon deduplication	869
Parameter troubleshooting	870
Ad skipping troubleshooting	870
Symptoms	870
Skip reasons	870
NEW_CREATIVE skipping	871
ADS timeout skipping	873
VAST parsing skipping	874
Duration mismatch skipping	875
Session variable skipping	876
Monitoring ad skipping	877
Prevention best practices	878
Reference materials	879
MediaTailor resources	882
Document history	884

What is AWS Elemental MediaTailor?

AWS Elemental MediaTailor is a scalable ad insertion and channel assembly service that runs in the AWS Cloud. With MediaTailor, you can serve targeted ad content to viewers and create linear streams while maintaining broadcast quality in over-the-top (OTT) video applications. MediaTailor ad insertion supports Apple HTTP Live Streaming (HLS) and MPEG Dynamic Adaptive Streaming over HTTP (DASH) for video on demand (VOD) and live workflows.

AWS Elemental MediaTailor ad insertion offers important advances over traditional ad-tracking systems: ads are better monetized, more consistent in video quality and resolution, and easier to manage across multi-platform environments. MediaTailor simplifies your ad workflow by allowing all IP-connected devices to render ads in the same way as they render other content. The service also offers advanced tracking of ad views, which further increases the monetization of content.

AWS Elemental MediaTailor channel assembly is a manifest-only service that allows you to create linear streaming channels using your existing video on demand (VOD) content. MediaTailor never touches your content segments, which are served directly from your origin server. Instead, MediaTailor fetches the manifests from your origin, and uses them to assemble a live sliding manifest window that references the underlying content segments.

MediaTailor channel assembly makes it easy to monetize your channel by inserting ad breaks into your stream without having to condition it with SCTE-35 markers. You can use channel assembly with MediaTailor ad insertion, or another server-side ad insertion service.

Origin server requirements

AWS Elemental MediaTailor has specific requirements for origin server communication:

- **Supported ports** - MediaTailor only accepts origins using standard HTTP and HTTPS ports:
 - Port 80 for HTTP connections
 - Port 443 for HTTPS connections

MediaTailor does not support custom ports for origin server communication.

- **Protocol requirements** - For secure communication, MediaTailor requires HTTPS for certain origin types and authentication scenarios. For more information, see [the section called "Integrating a content source"](#).

MediaTailor concepts

Here's an overview of the concepts that are used throughout the *AWS Elemental MediaTailor User Guide*.

HLS playlist types

HTTP Live Streaming (HLS) uses two primary types of playlists: multivariant playlists and media playlists. Understanding the difference between these playlist types is essential for configuring and troubleshooting MediaTailor workflows.

Multivariant playlist

A multivariant playlist is the top-level index file in an HLS streaming workflow that lists all available renditions of the content. It contains references to media playlists but does not contain any media segments itself. The multivariant playlist allows players to select the most appropriate rendition based on network conditions, device capabilities, or user preferences.

This playlist type is also known by several other names in various contexts:

- Master playlist (deprecated term)
- Master manifest (common industry term)
- Primary playlist
- Main playlist
- Index file
- Master M3U8

In MediaTailor workflows, the multivariant playlist is the entry point for playback requests and is where ad personalization begins.

Example Multivariant playlist example

```
#EXTM3U
#EXT-X-VERSION:3
#EXT-X-INDEPENDENT-SEGMENTS
#EXT-X-STREAM-INF:BANDWIDTH=2665726,AVERAGE-
BANDWIDTH=2526299,RESOLUTION=960x540,CODECS="avc1.640029,mp4a.40.2",SUBTITLES="subtitles"
index_1.m3u8
#EXT-X-STREAM-INF:BANDWIDTH=4786126,AVERAGE-
BANDWIDTH=4547599,RESOLUTION=1280x720,CODECS="avc1.640029,mp4a.40.2",SUBTITLES="subtitles"
```

```
index_2.m3u8
#EXT-X-STREAM-INF:BANDWIDTH=8171787,AVERAGE-
BANDWIDTH=7768157,RESOLUTION=1920x1080,CODECS="avc1.640029,mp4a.40.2",SUBTITLES="subtitles"
index_3.m3u8
```

Media playlist

A media playlist contains the actual media segment information for a specific rendition (quality level) of the content. It includes timing information, segment URLs, and other metadata required for playback of a single rendition. Each media playlist represents a different combination of resolution, bitrate, and other characteristics.

This playlist type is also known by several other names:

- Variant playlist
- Child manifest (common industry term)
- Chunklist
- Media M3U8
- Rendition playlist

In MediaTailor workflows, media playlists are personalized to include both content segments and ad segments in the proper sequence.

Example Media playlist example

```
#EXTM3U
#EXT-X-VERSION:6
#EXT-X-TARGETDURATION:6
#EXT-X-MEDIA-SEQUENCE:123456
#EXT-X-DISCONTINUITY-SEQUENCE:0
#EXTINF:6.0,
segment123456.ts
#EXTINF:6.0,
segment123457.ts
#EXT-X-DISCONTINUITY
#EXTINF:5.0,
ad-segment1.ts
#EXTINF:5.0,
ad-segment2.ts
#EXT-X-DISCONTINUITY
#EXTINF:6.0,
segment123458.ts
```

Note

Both multivariant and media playlists are text-based files that use the .m3u8 extension. The terminology for these files varies across different documentation and contexts, but the fundamental roles remain the same in the HLS streaming architecture.

When configuring MediaTailor, you typically provide the URL to the multivariant playlist in your content origin. MediaTailor then handles the personalization of both the multivariant playlist and the media playlists to insert ads according to your configuration.

For more information about HLS playlist specifications, see the [HTTP Live Streaming RFC](#).

DASH manifest types

Dynamic Adaptive Streaming over HTTP (DASH) uses a Media Presentation Description (MPD) manifest to deliver streaming content. Understanding the structure and components of DASH manifests is essential for configuring and troubleshooting MediaTailor workflows.

MPD (Media Presentation Description)

The MPD is the primary manifest file in DASH streaming that describes the structure and availability of media content. It contains information about periods, adaptation sets, representations, and segments that make up the streaming content.

This manifest type is also known by several other names in various contexts:

- DASH manifest
- DASH MPD
- Master manifest (when comparing to HLS)
- Presentation manifest

In MediaTailor workflows, the MPD is the entry point for playback requests and is where ad personalization begins.

Example MPD manifest example

```
<?xml version="1.0" encoding="UTF-8"?>
<MPD xmlns="urn:mpeg:dash:schema:mpd:2011" xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance" xsi:schemaLocation="urn:mpeg:dash:schema:mpd:2011
```

```

DASH-MPD.xsd" profiles="urn:mpeg:dash:profile:isoff-live:2011"
type="dynamic" minBufferTime="PT5.000S" maxSegmentDuration="PT2.005S"
availabilityStartTime="2020-01-01T00:00:00Z" publishTime="2020-01-01T12:30:00Z"
minimumUpdatePeriod="PT2.000S" timeShiftBufferDepth="PT5M">
  <Period id="1" start="PT0.000S">
    <AdaptationSet id="1" contentType="video" segmentAlignment="true"
bitstreamSwitching="true" frameRate="30000/1001" maxWidth="1920" maxHeight="1080"
par="16:9">
      <Representation id="1" mimeType="video/mp4" codecs="avc1.640028" width="1920"
height="1080" bandwidth="5000000">
        <SegmentTemplate timescale="90000" initialization="init-stream
$RepresentationID$.m4s" media="chunk-stream$RepresentationID$-$Number%05d$.m4s"
startNumber="1" duration="180000"/>
      </Representation>
      <Representation id="2" mimeType="video/mp4" codecs="avc1.4d401f" width="1280"
height="720" bandwidth="2800000">
        <SegmentTemplate timescale="90000" initialization="init-stream
$RepresentationID$.m4s" media="chunk-stream$RepresentationID$-$Number%05d$.m4s"
startNumber="1" duration="180000"/>
      </Representation>
      <Representation id="3" mimeType="video/mp4" codecs="avc1.4d401e" width="640"
height="360" bandwidth="1100000">
        <SegmentTemplate timescale="90000" initialization="init-stream
$RepresentationID$.m4s" media="chunk-stream$RepresentationID$-$Number%05d$.m4s"
startNumber="1" duration="180000"/>
      </Representation>
    </AdaptationSet>
    <AdaptationSet id="2" contentType="audio" segmentAlignment="true" lang="eng">
      <Representation id="4" mimeType="audio/mp4" codecs="mp4a.40.2"
audioSamplingRate="48000" bandwidth="128000">
        <SegmentTemplate timescale="48000" initialization="init-stream
$RepresentationID$.m4s" media="chunk-stream$RepresentationID$-$Number%05d$.m4s"
startNumber="1" duration="96000"/>
      </Representation>
    </AdaptationSet>
  </Period>
</MPD>

```

Period

A Period is a temporal section of a DASH presentation. Each Period contains one or more adaptation sets and represents a span of media time. In ad insertion workflows, separate Periods are typically used to delineate between content and ads.

This component is also known by several other names:

- Content segment
- Timeline section
- Program segment

In MediaTailor workflows, Periods are used to separate main content from ad content, with each ad typically represented by its own Period.

Example Period example

```
<Period id="ad-period-1" start="PT30.000S" duration="PT15.000S">
  <AdaptationSet id="1" contentType="video" segmentAlignment="true"
    bitstreamSwitching="true" frameRate="30000/1001" maxWidth="1920" maxHeight="1080"
    par="16:9">
    <Representation id="1" mimeType="video/mp4" codecs="avc1.640028" width="1920"
      height="1080" bandwidth="5000000">
      <SegmentTemplate timescale="90000" initialization="ad1/init-stream
$RepresentationID$.m4s" media="ad1/chunk-stream$RepresentationID$-$Number%05d$.m4s"
      startNumber="1" duration="180000"/>
    </Representation>
  </AdaptationSet>
</Period>
```

AdaptationSet

An AdaptationSet groups a set of interchangeable encoded versions of one or several media content components. For example, one AdaptationSet might contain multiple video quality levels, while another might contain multiple audio language options.

This component is also known as:

- Media component group
- Stream set
- Track group

In MediaTailor workflows, AdaptationSets are preserved during ad insertion to maintain consistent media types between content and ads.

Representation

A Representation is a specific encoded version of the media content within an AdaptationSet. Each Representation typically differs in bitrate, resolution, or other encoding parameters,

allowing clients to select the most appropriate version based on network conditions and device capabilities.

This component is also known as:

- Rendition (similar to HLS)
- Quality level
- Bitrate variant
- Stream variant

In MediaTailor workflows, Representations in ad Periods are matched as closely as possible to the Representations in content Periods to ensure a smooth viewing experience.

Segment

A Segment is a unit of media data that can be referenced individually by a URL. Segments contain the actual media content (video, audio, etc.) and are referenced within the MPD. There are two main types of segments in DASH:

- **Initialization Segment:** Contains initialization information for a Representation, such as codec parameters and timing information.
- **Media Segment:** Contains the actual media data for a specific time range within a Representation.

In MediaTailor workflows, segment URLs are often modified to point to the appropriate content or ad media files.

Note

DASH manifests are XML-based files that typically use the .mpd extension. The terminology for these files and their components may vary across different documentation and contexts, but the fundamental structure remains the same in the DASH streaming architecture.

When configuring MediaTailor, you provide the URL to the MPD manifest in your content origin. MediaTailor then handles the personalization of the manifest, typically by inserting additional Periods for ads according to your configuration.

For more information about DASH manifest specifications, see the [DASH Industry Forum Implementation Guidelines](#).

Advanced DASH manifest concepts

When working with DASH manifests in MediaTailor, understanding the following advanced concepts can help you configure and troubleshoot your streaming workflows more effectively:

Period start time calculation

In DASH manifests, Period start times are calculated based on the MPD's `availabilityStartTime` attribute and the Period's `start` attribute. For live streams, the start time is relative to the availability start time, while for VOD content, it's typically relative to the beginning of the presentation.

When MediaTailor inserts ad Periods, it carefully calculates the start times to ensure seamless transitions between content and ads. This calculation takes into account:

- The original Period's start time
- The duration of preceding ad Periods
- Any time offset specified in the ad decision server response

Preroll timing calculations

Preroll ads in DASH manifests require special handling because they appear before the main content begins. MediaTailor inserts preroll ads as separate Periods at the beginning of the manifest with appropriate start times and durations.

For preroll ads, MediaTailor:

- Creates a new Period for each preroll ad
- Sets the first preroll Period's start time to 0
- Adjusts the start time of the main content Period to account for the total duration of all preroll ads

Live-to-VOD transitions

DASH manifests can transition from live to VOD (Video on Demand) format when a live stream ends. This transition involves changing the MPD's `type` attribute from `dynamic` to `static` and adjusting other attributes like `timeShiftBufferDepth` and `minimumUpdatePeriod`.

When MediaTailor processes manifests during live-to-VOD transitions, it ensures that:

- Ad markers are preserved in the VOD manifest
- Period start times are adjusted to maintain proper timing
- The manifest remains compatible with VOD playback clients

DRM handling

Digital Rights Management (DRM) information in DASH manifests is typically included in the ContentProtection elements within AdaptationSets or Representations. MediaTailor preserves these elements during ad insertion to ensure that content protection remains intact.

When processing DRM-protected content, MediaTailor:

- Maintains all ContentProtection elements from the original manifest
- Ensures that ad content uses compatible DRM schemes if applicable
- Preserves any DRM-related attributes and elements throughout the manifest

SCC flags

Supplemental Content Control (SCC) flags in DASH manifests provide additional information about content characteristics and playback requirements. These flags are typically included as attributes or elements within the MPD structure.

Common SCC flags that MediaTailor processes include:

- Content rating information
- Accessibility features (closed captions, audio descriptions)
- Content advisory notices
- Playback restrictions

MediaTailor preserves these flags during manifest processing to ensure that all content metadata is maintained in the personalized manifest.

Understanding these advanced concepts helps you configure MediaTailor for optimal performance and troubleshoot any issues that may arise in your DASH streaming workflows.

Ad insertion concepts

Here's an overview of the concepts that are related to ad insertion.

Ad avail

A specific unit of advertising time within an ad break that can be sold to advertisers. An ad break may contain multiple ad avails. When MediaTailor receives a VAST response from an ad decision server, it fills these avails with personalized ads.

Ad break

The time period during programming when commercials are shown. Ad breaks can occur before content (pre-roll), during content (mid-roll), or after content (post-roll). MediaTailor identifies ad breaks in manifests through markers like SCTE-35.

Ad decision server (ADS)

A server that provides advertising spot specifications based on criteria including current advertising campaigns and viewer preferences.

Configuration

An object in MediaTailor that you interact with. The configuration holds location information about the origin server and the ad decision server (ADS). The configuration also holds endpoints that provide access points in and out of MediaTailor.

Dynamic transcoding

A process that matches the ad quality and format to the primary video content when content is requested. Dynamic transcoding reduces storage requirements and ensures that playback seamlessly transitions between the ad and video content.

Manifest manipulation

The process of rewriting manifests from the origin server so that the manifests reference the appropriate ad and content fragments. Ads are determined by the VAST response from the ad decision server (ADS). As playback progresses, MediaTailor performs ad insertion or ad replacement into the content stream.

VAST and VMAP

Video Ad Serving Template (VAST) and Video Multiple Ad Playlist (VMAP) are XML responses that the ad decision server sends to ad requests from MediaTailor. The responses dictate what ads MediaTailor inserts in the manifest. VMAP also includes timing for ad breaks and the ad avails within them. For more information about the logic behind MediaTailor ad insertion, see [Understanding AWS Elemental MediaTailor ad insertion behavior](#). For more information about how MediaTailor works with VAST, see [the section called “Ad server integration requirements”](#).

Channel assembly concepts

Here's an overview of the concepts that are related to channel assembly.

Channels

A channel assembles your source manifests into a linear stream. Each channel has one or more outputs that contain playback URLs accessed by players. The channel outputs correspond to the package configuration settings you create for your VOD sources. A channel contains a schedule, which determines when VOD sources will play in the channel's stream.

Package configuration

A packager configuration is a representation of your VOD source that contains specific packaged format characteristics. You associate your package configurations with channel outputs to create playback streams for your VOD source's packaged formats, such as HTTP Live Streaming (HLS).

Schedule

Each channel is made up of programs that are arranged into the channel's schedule. The schedule determines what time the programs will play in the channel's linear stream.

Source locations

A source location represents the origin server where your assets are stored. It can be Amazon S3, an HTTP server, a Content Delivery Network (CDN), or a packaging infrastructure such as MediaPackage.

VOD sources

A VOD source represents a single piece of content, such as a movie or an episode of a TV show. You associate VOD sources with programs to add them to your channel's linear stream.

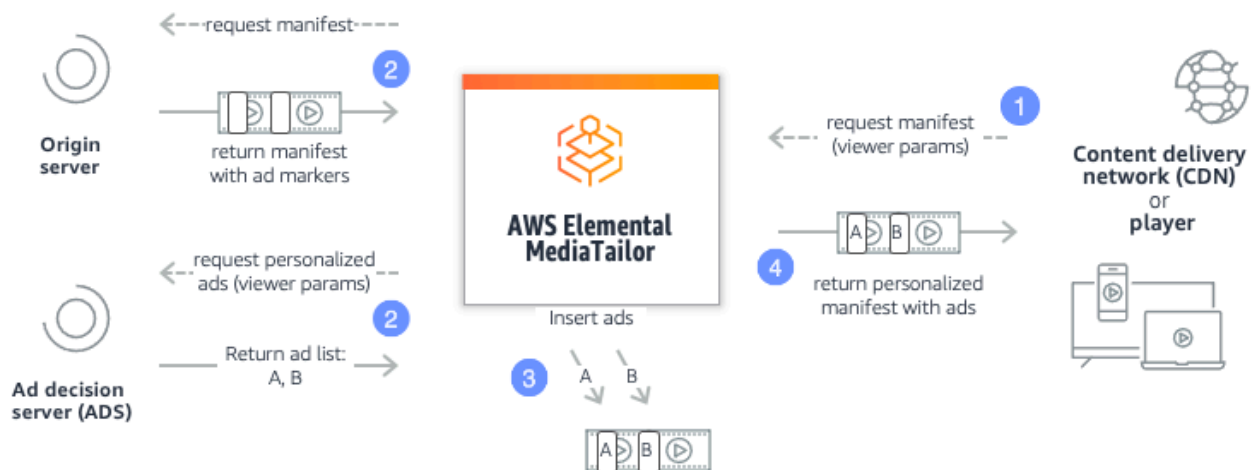
Audience

An audience defines a viewer cohort which can optionally have alternate content. You can define audiences on standard linear channels.

How MediaTailor ad insertion works

AWS Elemental MediaTailor interacts between your content delivery network (CDN), origin server, and ad decision server (ADS) to stitch personalized ads into ad breaks within live and video on demand content.

Here's an overview of how MediaTailor ad insertion works:



1. A player or CDN such as Amazon CloudFront sends a request to MediaTailor for HLS or DASH content. The request contains parameters from the player with information about the viewer, which is used for ad personalization.
2. MediaTailor sends a request to the ADS that contains the viewer information. The ADS chooses ads based on the viewer information and current ad campaigns. It returns the URLs to the ad creatives in a VAST or VMAP response to MediaTailor.

If you pre-conditioned the ads, the URLs point to the pre-transcoded ads. For information about ad stitching with pre-transcoded ads, see [Preconditioned ads](#).

3. MediaTailor manipulates the manifest to include the ad URLs returned from the ADS, transcoded to match the encoding characteristics of the origin content. If you're using preconditioned ads, it's your responsibility to ensure that the ad matches the template manifest.

If an ad hasn't yet been transcoded to match the content, MediaTailor will skip inserting it and use MediaConvert to prepare the ad so that it's ready for the next request.

4. MediaTailor returns the fully personalized manifest to the requesting CDN or player.

The ADS tracks the ads viewed based on viewing milestones such as start of ad, middle of ad, and end of ad. As playback progresses, the player or MediaTailor sends ad tracking beacons to the ADS ad tracking URL, to record how much of an ad has been viewed. In the session initialization with MediaTailor, the player indicates whether it or MediaTailor is to send these beacons for the session.

For information about how to get started with ad insertion, see [Getting started with MediaTailor](#).

AWS Elemental MediaTailor ad insertion event flow

AWS Elemental MediaTailor processes manifest personalization for server-side ad insertion through a predictable sequence of events. Understanding this event flow helps you comprehend how MediaTailor transforms ad opportunities into personalized viewing experiences and provides a foundation for troubleshooting when issues occur.

Each ad insertion opportunity follows a chronological sequence of events that MediaTailor logs for visibility and monitoring. These events represent key milestones in the manifest personalization process, from detecting an ad opportunity to delivering tracking information.

Standard event sequence

When MediaTailor successfully processes an ad insertion opportunity, the following sequence of events typically occurs:

1. **Ad opportunity detection** - MediaTailor detects an ad marker (such as SCTE-35) in the content manifest, indicating an ad personalization opportunity is available.
2. **Ad decision server request** - MediaTailor sends a request to the configured ad decision server (ADS) to retrieve advertisements for the detected opportunity, including viewer parameters and ad break duration.
3. **Ad response processing** - MediaTailor receives a response from the ADS containing ad creative information, tracking URLs, and metadata about the advertisements to be included. For more information about VAST, VMAP, and VPAID response formats, see [Ad server integration requirements](#).
4. **Manifest personalization** - MediaTailor successfully processes the ad response, transcodes the advertisements if necessary, and generates a personalized manifest that includes relative URLs for both the ad segments and origin content. MediaTailor then sends the personalized manifest to the playback device.

5. Tracking beacon activation - A tracking beacon is fired to report ad events (such as impressions, quartiles, and completion) back to the ad server or other measurement systems. In server-side reporting mode (default), MediaTailor fires the beacon based on player segment requests. In client-side reporting mode, the playback device fires the beacon using URLs provided in the personalized manifest.

Typical timing: This entire sequence usually completes within 2-5 seconds, depending on ad decision server response time and ad creative processing requirements.

Common event variations

Not all ad insertion attempts follow the standard successful sequence. MediaTailor handles various scenarios that can alter the event flow:

Empty ad response scenario

When the ad decision server returns no advertisements:

1. Ad opportunity detection occurs normally
2. Ad decision server request is sent successfully
3. ADS returns an empty response with zero ads
4. No ads are included in the personalized manifest for this opportunity

This is a normal scenario that can occur due to ad inventory availability, targeting criteria, or business rules configured in the ad decision server. In this case, the underlying content will be displayed unless the stream is a live or live-to-VOD stream and slate has been configured. For more information, see [MediaTailor slate ad insertion](#).

Error scenarios

When technical issues prevent successful ad insertion, the event flow might be interrupted at various points:

- **Ad decision server timeout** - The request to the ADS exceeds the configured timeout threshold, preventing ad retrieval.
- **Communication errors** - Network or connectivity issues prevent MediaTailor from reaching the ad decision server.

- **Response parsing errors** - The ADS returns a response that MediaTailor cannot parse due to invalid VAST format or structure.

In error scenarios, MediaTailor typically continues with content playback without ads, ensuring uninterrupted viewer experience. For live or live-to-VOD streams with slate configured, slate content might be displayed instead. For more information, see [MediaTailor slate ad insertion](#).

VAST redirect scenarios

When the initial VAST response contains a redirect to another ad server:

1. Ad opportunity detection and initial ADS request occur normally
2. Initial response contains a redirect instruction
3. MediaTailor follows the redirect to retrieve the final ad response
4. Manifest personalization proceeds with the final ad content
5. Tracking beacon activation occurs normally

VAST redirects are common in programmatic advertising and allow for ad server chaining and real-time decisioning. MediaTailor allows up to 7 wrapper redirects and unlimited HTTP redirects during the ad retrieval process.

Event timing overview

Understanding the timing characteristics of MediaTailor event flows helps set appropriate expectations for ad insertion performance:

- **Ad opportunity detection** - Occurs when MediaTailor receives a GetManifest request from the player and encounters the ad markers in the content manifest.
- **Ad decision server interaction** - Typically takes 100-500 milliseconds, depending on ADS response time and network conditions.
- **Manifest personalization** - Usually completes within 50-200 milliseconds after receiving the ADS response.
- **Tracking beacon timing** - Varies by reporting mode. For server-side beacons, timing is based on segment requests from the client player; client-side beacons fire based on player implementation.

Performance considerations: The total time from ad opportunity detection to manifest delivery should typically remain under 5 seconds to maintain optimal viewer experience. Individual components complete in milliseconds as described above. However, the 5-second threshold accounts for potential timeout values, retry attempts, and network variability that can occur during the complete ad insertion workflow. Longer delays might indicate ADS performance issues or network connectivity problems.

Using event flow knowledge

Understanding MediaTailor event flow provides the foundation for:

- **Implementation planning** - Knowing the event sequence helps in designing player integration and ad server configuration.
- **Performance optimization** - Understanding timing expectations enables identification of bottlenecks and optimization opportunities.
- **Troubleshooting preparation** - Familiarity with normal event flows makes it easier to identify when something goes wrong.

For detailed troubleshooting guidance using event flow analysis, see [Troubleshooting](#). For technical details about event logging and monitoring, see [Viewing logs](#).

Related services

- **Amazon CloudFront** is a global content delivery network (CDN) service that securely delivers data and videos to your viewers. Use CloudFront to deliver content with the best possible performance. For more information about CloudFront, see the [Amazon CloudFront website](#).
- **AWS Elemental MediaPackage** is a just-in-time packaging and origination service that customizes live video assets for distribution in a format that is compatible with the device that makes the request. Use AWS Elemental MediaPackage as an origin server to prepare content and add ad markers before sending streams to MediaTailor. For more information about how MediaTailor works with origin servers, see [How MediaTailor ad insertion works](#).
- **AWS Identity and Access Management (IAM)** is a web service that helps you securely control access to AWS resources for your users. Use IAM to control who can use your AWS resources (authentication) and what resources they can use in which ways (authorization). For more information, see [Setting up AWS Elemental MediaTailor](#).

Accessing MediaTailor

You can access MediaTailor using the service's console.

Access your AWS account by providing credentials that verify that you have permissions to use the services.

To log in to the MediaTailor console, use the following link: **`https://console.aws.amazon.com/mediatailor/home`**.

Pricing for MediaTailor

As with other AWS products, there are no contracts or minimum commitments for using MediaTailor. You are charged based on your use of the service. For more information, see [MediaTailor pricing](#).

Regions for MediaTailor

To reduce data latency in your applications, MediaTailor offers regional endpoints to make your requests. To view the list of Regions in which MediaTailor is available, see [Regional endpoints](#).

Setting up AWS Elemental MediaTailor

This section guides you through the steps required to configure users to access AWS Elemental MediaTailor. For background and additional information about identity and access management for MediaTailor, see [Identity and Access Management for AWS Elemental MediaTailor](#).

To start using AWS Elemental MediaTailor, complete the following steps.

Topics

- [Sign up for an AWS account](#)

Sign up for an AWS account

To get started with AWS, you need an AWS account. For information about creating an AWS account, see [Getting started with an AWS account](#) in the *AWS Account Management Reference Guide*.

Getting started with MediaTailor

To get started with MediaTailor, you can choose between two tutorials—one on setting up ad insertion, and another on channel assembly. You can also explore a hands-on workshop for live events. The ad insertion tutorial will guide you through sending a playback request to MediaTailor to incorporate personalized ads into your content stream. The channel assembly tutorial will show you how to view your channel's stream, including the personalized ads, directly in a browser using a playback URL.

You can also explore the [Live Event Framework Workshop](#). This hands-on workshop guides you through the complete lifecycle of an AWS Elemental MediaTailor live event setup, from initial discovery and deployment to optimization and scaling. You can follow the workshop for a simple configuration that uses one MediaTailor setup for all events and channels, or for an advanced architecture that uses dedicated configurations per event or channel.

Topics

- [Getting started with MediaTailor ad insertion](#)
- [Getting started with MediaTailor channel assembly](#)

Getting started with MediaTailor ad insertion

To use AWS Elemental MediaTailor, you need an AWS account and permissions to access, view, and edit MediaTailor configurations. For information on how to do this, see [Setting up AWS Elemental MediaTailor](#).

This Getting Started tutorial shows you how to perform the following tasks:

- Prepare your HLS or DASH content streams
- Configure an ad decision server (ADS) template URL
- Create a MediaTailor configuration that contains a playback endpoint
- Use your player or content delivery network (CDN) to make a playback request to MediaTailor

For advanced ADS URL configuration using dynamic variables and player parameters, see [MediaTailor dynamic ad variables for ADS requests](#). For information about passing parameters through CDNs, see [MediaTailor manifest query parameters](#).

When you're finished, you'll be able to send a playback request to MediaTailor for personalized ad content in your stream.

Topics

- [Prerequisites](#)
- [Step 1: Access AWS Elemental MediaTailor](#)
- [Step 2: Prepare a stream](#)
- [Step 3: Configure ADS request URL and query parameters](#)
- [Step 4: Create a configuration](#)
- [Step 5: Test the configuration](#)
- [Step 6: Send the playback request to AWS Elemental MediaTailor](#)
- [Step 7 \(optional\): Monitor AWS Elemental MediaTailor activity](#)
- [Step 8: Clean up](#)

Prerequisites

Before you begin, be sure that you've completed the steps in [Setting up AWS Elemental MediaTailor](#).

Step 1: Access AWS Elemental MediaTailor

Using your IAM credentials, sign in to the MediaTailor console at **`https://console.aws.amazon.com/mediatailor/home`**.

Step 2: Prepare a stream

Configure your origin server to produce manifests for HLS or DASH that are compatible with AWS Elemental MediaTailor. Your origin server must use standard HTTP ports (port 80) or HTTPS ports (port 443). MediaTailor does not support custom ports for origin server communication.

Prepare an HLS stream

HLS manifests must satisfy the following requirements:

- Manifests must be accessible on the public internet.

- Manifests must be live or video on demand (VOD).
- Manifests must have an EXT-X-VERSION of 3 or higher.
- For live content, manifests must contain markers to delineate ad avails. This is optional for VOD content, which can use VMAP timeoffsets instead.

The manifest file must have ad slots marked with one of the following:

- **#EXT-X-CUE-OUT / #EXT-X-CUE-IN** (more common) with durations as shown in the following example.

```
#EXT-X-CUE-OUT:60.00  
#EXT-X-CUE-IN
```

- **#EXT-X-DATERANGE** (less common) with durations as shown in the following example.

```
#EXT-X-DATERANGE:ID="",START-DATE="",DURATION=30.000,SCTE35-OUT=0xF  
#EXT-X-DATERANGE:ID="",START-DATE="",DURATION=30.000,SCTE35-OUT=0xF
```

All fields shown for #EXT-X-DATERANGE are required.

The way that you configure the ad markers in the manifest influences whether ads are inserted in a stream or replace other fragments in the stream. For more information, see [the section called “Understanding ad insertion behavior”](#).

- HLS multivariant playlists must follow the HLS specification documented at [HTTP live streaming: Master playlist tags](#). In particular, #EXT-X-STREAM-INF must include the fields RESOLUTION, BANDWIDTH, and CODEC.

After you have configured the stream, note the content origin URL prefix for the multivariant playlist. You need it to create the configuration in AWS Elemental MediaTailor, later in this tutorial.

Prepare a DASH stream

DASH manifests must satisfy the following requirements:

- Manifests must be accessible on the public internet.
- Manifests must be live or video on demand (VOD).
- Manifests must mark events as ad avails using either splice insert markers or time signal markers. You can provide the ad markers in clear XML or in base64-encoded binary. For splice insert, the out-of-network indicator must be enabled. For time signal markers, the segmentation type ID,

located inside the segmentation UPID, must be a cue-out value recognized by AWS Elemental MediaTailor. The ad avail starts at the event start and lasts for the event duration, if one is specified, or until the next event starts.

The following example shows an event designated as an ad avail using splice insert markers. The duration for this ad avail is the event's duration.

```
<Period start="PT444806.040S" id="123586" duration="PT15.000S">
  <EventStream timescale="90000" schemeIdUri="urn:scte:scte35:2013:xml">
    <Event duration="1350000">
      <scte35:SpliceInfoSection protocolVersion="0" ptsAdjustment="180832"
tier="4095">
        <scte35:SpliceInsert spliceEventId="4026531855"
spliceEventCancelIndicator="false" outOfNetworkIndicator="true"
spliceImmediateFlag="false" uniqueProgramId="1" availNum="1" availsExpected="1">
          <scte35:Program><scte35:SpliceTime ptsTime="5672624400"/></
scte35:Program>
            <scte35:BreakDuration autoReturn="true" duration="1350000"/>
          </scte35:SpliceInsert>
        </scte35:SpliceInfoSection>
      </Event>
    </EventStream>
    <AdaptationSet mimeType="video/mp4"
      ...
    </AdaptationSet>
  </Period>
```

- Ad avails must have the same AdaptationSet and Representation settings as content streams. AWS Elemental MediaTailor uses these settings to transcode the ads to match the content stream, for smooth switching between the two.

After you configure the stream, note the content origin URL prefix for the DASH manifest. You need it to create the configuration in AWS Elemental MediaTailor, later in this tutorial.

Step 3: Configure ADS request URL and query parameters

To determine the query parameters that the ADS requires, generate an ad tag URL from the ADS. This URL acts as a template for requests to the ADS, and consists of the following:

- Static values

- Values generated by AWS Elemental MediaTailor (denoted by `session` or `avail` query parameters)
- Values generated by players, obtained from the client application (denoted by `player_params` query parameters)

Example Ad tag URL from an ADS

```
https://my.ads.com/ad?
output=vast&content_id=12345678&playerSession=[session.id]&cust_params=[player_params.cust_params]
```

Where:

- **output** and **content_id** are static values
- **playerSession=[session.id]** is a dynamic value provided by AWS Elemental MediaTailor. The value of **[session.id]** changes for each player session and results in a different URL for the VAST request for each session.
- **cust_params** are player-supplied dynamic values

The multivariant playlist request from the player must provide key-value pairs that correspond to the `player_params` query parameters in the ADS request URL. For more information about configuring key-value pairs in the request to AWS Elemental MediaTailor, see [MediaTailor dynamic ad variables for ADS requests](#).

Enter the configured "template" URL when you create the origin server/ADS mapping in MediaTailor, in [Step 4: Create a configuration](#).

Testing

You can use a static VAST response from your ADS for testing purposes. Ideally, the VAST response returns a mezzanine quality MP4 rendition that AWS Elemental MediaTailor can transcode. If the response from the ADS contains multiple playback renditions, MediaTailor picks the highest quality and resolution MP4 rendition and sends it to the transcoder.

Step 4: Create a configuration

The AWS Elemental MediaTailor configuration holds mapping information for the origin server and ADS.

To create a configuration (console)

1. Open the MediaTailor console at <https://console.aws.amazon.com/mediatailor/>.
2. On the **Configurations** page, choose **Create configuration**.
3. In the **Configuration** section at the bottom of the page, for **Configuration name**, enter a unique name that describes the configuration. The name is the primary identifier for the configuration. The maximum length allowed is 512 characters.
4. For **Video content source**, enter the URL prefix for the HLS multivariant playlist or DASH manifest for this stream, minus the asset ID. For example, if the multivariant playlist URL is `http://origin-server.com/a/master.m3u8`, you would enter `http://origin-server.com/a/`. Alternatively, you can enter a shorter prefix such as `http://origin-server.com`, but then you must include the `/a/` in the asset ID in the player request for content. The maximum length is 512 characters.

Note

If your content origin uses HTTPS, its certificate must be from a well-known certificate authority. (It can't be a self-signed certificate.) Otherwise, AWS Elemental MediaTailor fails to connect to the content origin and can't serve manifests in response to player requests.

5. For **Ad decision server**, enter the URL for your ADS. This is either the URL with variables as described in [Step 3: Configure ADS request URL and query parameters](#), or the static VAST URL that you are using for testing purposes. The maximum length is 25,000 characters.

Note

If your ADS uses HTTPS, its certificate must be from a well-known certificate authority. (It can't be a self-signed certificate.) The same is true for mezzanine ad URLs returned by the ADS. Otherwise, MediaTailor fails to retrieve and stitch ads into the manifests from the content origin.

6. (Optional as needed for DASH) For **Location**, choose **DISABLED** if you have CDN routing rules set up for accessing MediaTailor manifests and you are either using client-side reporting or your players support sticky HTTP redirects.

For more information about the **Location** feature, see [the section called "Location feature"](#).

7. (Optional) If your origin server produces single-period DASH manifests, choose **DASH mpd manifest origin type**, and then choose **SINGLE_PERIOD**. By default, MediaTailor handles DASH manifests as multi-period manifests. For more information, see [the section called "Integrating an MPEG-DASH source"](#).
8. Choose **Create configuration**.

AWS Elemental MediaTailor displays the new configuration on the **Configurations** page.

Step 5: Test the configuration

After you save the configuration, test the stream using a URL in the appropriate format for your streaming protocol:

- Example: HLS

```
playback-endpoint/v1/master/hashed-account-id/origin-id/master.m3u8
```

- Example: DASH

```
playback-endpoint/v1/dash/hashed-account-id/origin-id/manifest.mpd
```

Where:

- `playback-endpoint` is the unique playback endpoint that AWS Elemental MediaTailor generated when the configuration was created.

Example

```
https://777788889999.mediatailor.us-east-1.amazonaws.com
```

- `hashed-account-id` is your AWS account ID.

Example

```
777788889999
```

- `origin-id` is the name that you gave when creating the configuration.

Example

```
myOrigin
```

- `master.m3u8` or `manifest.mpd` is the name of the manifest from the test stream plus its file extension. Define this so that you get a fully identified manifest when you append this to the video content source that you configured in [the section called “Step 4: Create a configuration”](#).

Using the values from the preceding examples, the full URLs are the following.

- Example: HLS

```
https://777788889999.mediatailor.us-east-1.amazonaws.com/v1/master/  
AKIAIOSFODNN7EXAMPLE/myOrigin/master.m3u8
```

- Example: DASH

```
https://777788889999.mediatailor.us-east-1.amazonaws.com/v1/dash/  
AKIAIOSFODNN7EXAMPLE/myOrigin/manifest.mpd
```

You can test the stream using one of the following methods.

- As shown in the preceding example, enter the URL in a standalone player.
- Test the stream in your own player environment.

Step 6: Send the playback request to AWS Elemental MediaTailor

Configure the downstream player or CDN to send playback requests to the configuration's playback endpoint provided from AWS Elemental MediaTailor. Any player-defined dynamic variables that you used in the ADS request URL in [Step 3: Configure ADS request URL and query parameters](#) must be defined in the manifest request from the player.

Example

Assume your template ADS URL is the following.

```
https://my.ads.com/ad?  
output=vast&content_id=12345678&playerSession=[session.id]&cust_params=[player_params.cust_params]
```

Then define `[player_params.cust_params]` in the player request by prefacing the key-value pair with `ads..` AWS Elemental MediaTailor passes parameters that aren't preceded with `ads.` to the origin server instead of the ADS.

The player request URL is some variation of the following HLS and DASH examples.

```
https://777788889999.mediatailor.us-east-1.amazonaws.com/v1/master/  
AKIAIOSFODNN7EXAMPLE/myOrigin/master.m3u8?ads.cust_params=viewerinfo
```

```
https://777788889999.mediatailor.us-east-1.amazonaws.com/v1/dash/AKIAIOSFODNN7EXAMPLE/  
myOrigin/manifest.mpd?ads.cust_params=viewerinfo
```

When AWS Elemental MediaTailor receives the player request, it defines the player variables based on the information in the request. The resulting ADS request URL is some variation of this.

```
https://my.ads.com/ad?  
output=vast&content_id=12345678&playerSession=<filled_in_session_id>&cust_params=viewerinfo
```

For more information about configuring key-value pairs to pass to the ADS, see [MediaTailor dynamic ad variables for ADS requests](#).

Step 7 (optional): Monitor AWS Elemental MediaTailor activity

Use Amazon CloudWatch and Amazon CloudWatch Logs to track AWS Elemental MediaTailor activity, such as the counts of requests, errors, and ad avails filled.

If this is your first time using CloudWatch with AWS Elemental MediaTailor, create an AWS Identity and Access Management (IAM) role to allow communication between the services.

To allow AWS Elemental MediaTailor access to CloudWatch (console)

1. Open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation pane of the IAM console, choose **Roles**, and then choose **Create role**.
3. Choose the **Another AWS account** role type.
4. For **Account ID**, enter your AWS account ID.
5. Select **Require external ID** and enter **midas**. This option automatically adds a condition to the trust policy that allows the service to assume the role only if the request includes the correct `sts:ExternalID`.

6. Choose **Next: Permissions**.
7. Add a permissions policy that specifies what actions this role can complete. Select from one of the following options, and then choose **Next: Review**:
 - **CloudWatchLogsFullAccess** to provide full access to Amazon CloudWatch Logs
 - **CloudWatchFullAccess** to provide full access to Amazon CloudWatch
8. For **Role name**, enter **MediaTailorLogger**, and then choose **Create role**.
9. On the **Roles** page, select the role that you just created.
10. Edit the trust relationship to update the principal:
 1. On the role's **Summary** page, choose the **Trust relationship** tab.
 2. Choose **Edit trust relationship**.
 3. In the policy document, change the principal to the AWS Elemental MediaTailor service. It should look like this.

```
"Principal": {  
    "Service": "mediatailor.amazonaws.com"  
},
```

The entire policy should read as follows.

4. Choose **Update Trust Policy**.

Step 8: Clean up

To avoid extraneous charges, delete all unnecessary configurations.

To delete a configuration (console)

1. On the AWS Elemental MediaTailor **Configurations** page, do one of the following:
 - Choose the **Configuration name** for the configuration that you want to delete.
 - In the **Configuration name** column, choose the radio button, and then choose **Delete**.
2. In the **Delete configuration** confirmation box, enter **Delete**, and then choose **Delete** again.

AWS Elemental MediaTailor removes the configuration.

Getting started with MediaTailor channel assembly

This Getting Started tutorial shows you how to perform the following tasks:

- Create a source location, and add source content to it
- Create a channel
- Create a program list to play your channel's content on a schedule
- Add personalized ads to the channel stream using AWS Elemental MediaTailor ad insertion

When you're finished, you'll be able to open a browser, enter the playback URL for your channel, and view your channel's stream containing personalized ads.

This tutorial walks you through the basic steps to get started with MediaTailor channel assembly. For more advanced information, see [Using AWS Elemental MediaTailor to create linear assembled streams](#).

Estimated cost

- The fee for an active channel is \$0.10 per hour. You aren't charged for channels that are inactive.

Topics

- [Prerequisites](#)
- [Step 1: Create a source location](#)
- [Step 2: Add VOD sources to your source location](#)
- [Step 3: Create a channel](#)
- [Step 4: Add programs to your channel's schedule](#)
- [Step 5 \(optional\): Use MediaTailor to insert personalized ads into your stream](#)
- [Step 6: Start your channel](#)
- [Step 7: Test your channel](#)
- [Step 8: Clean up](#)

Prerequisites

Before you begin this tutorial, you must complete these requirements:

- Be sure that you've completed the steps in [Setting up AWS Elemental MediaTailor](#).
- You must have assets available for both VOD source content and ad slate. You must know the path to the manifests for the assets.

Note

If you are using automated adaptive bitrate (ABR) or per title encoding, you must encode your assets so that all variants are the same length and have the same number of child tracks. We recommend that you use an encoding template with a minimum segment length of one second.

Step 1: Create a source location

A source location represents the origin server where your content is stored. It can be Amazon S3, a standard web server, a content delivery network (CDN), or a packaging origin, such as AWS Elemental MediaPackage.

MediaTailor fetches the content manifests from your source location, and uses them to assemble a live sliding manifest window that references the underlying content segments.

To create a source location, perform the following procedure.

To create a source location

1. Open the MediaTailor console at <https://console.aws.amazon.com/mediatailor/>.
2. In the navigation pane, choose **Channel assembly** > **Source locations**.
3. On the navigation bar, choose **Create source location**.
4. Under **Source location configuration**, enter an identifier and the location of your source content:
 - **Name:** An identifier for your source location, such as **my-origin**.
 - **Base URL:** The base URL of the origin server where your content is hosted, such as **https://111111111111.cloudfront.net**. The URL must be in a standard HTTP URL format, prefixed with **http://** or **https://**.
5. Choose **Create source location**.

Step 2: Add VOD sources to your source location

Now that you've defined one or more source locations for your channel, you can add one or more *VOD sources*. Each VOD source represents a single piece of content, such as a single movie, an episode of a TV show, or a highlight clip.

You must create at least one *package configuration* for your VOD source. Each package configuration contains the packaged format and manifest settings for your VOD sources. You then add your package configurations to your channel to create outputs.

You can use multiple package configurations to create different channel outputs. For example, if your VOD source is packaged as both HLS and DASH, you can create two package configurations for each format. You can then use the package configuration's source groups to create two channel outputs: one for HLS, one for DASH.

To add VOD sources and create package configurations

1. Open the MediaTailor console at <https://console.aws.amazon.com/mediatailor/>.
2. In the navigation pane, choose **Channel assembly** > **Source locations**.
3. In the **Source locations** pane, choose the source location that you created in the [To create a source location](#) procedure.
4. Choose **Add VOD source**.
5. Under **VOD source details**, enter a **Name** for your VOD source, such as **my-example-video**.
6. Under **Package configurations** > *source-group-name* enter information about the package configuration:

Note

Your source's package configurations must all have the same duration, as determined by the source's manifest. And, all of the sources within a package configuration must have the same number of child streams. To meet these requirements, we recommend that you use an encoding template for your assets. We recommend that you use an encoding template with a minimum segment length of one second. MediaTailor doesn't support per title or automated adaptive bitrate streaming (ABR) because these encoding methods violate these requirements.

- **Source group:** Enter a source group name that describes this package configuration, such as HLS-4k. Make a note of this name; you'll reference it when you create your channel's output. For more information, see [Using source groups with your channel's outputs](#).
 - **Type:** Select the packaged format for this configuration. MediaTailor supports HLS and DASH.
 - **Relative path:** The relative path from the source location's **Base HTTP URL** to the manifest. For example, `/my/path/index.m3u8`.
7. Choose **Add source**.
 8. Repeat steps 4-7 in this procedure to add the VOD source for your ad slate.

Step 3: Create a channel

A channel assembles your sources into a live linear stream. Each channel contains one or more outputs that correspond to your VOD source's package configurations.

First you create a channel, then you add your VOD sources to the channel's schedule by creating programs.

To create a channel

1. Open the MediaTailor console at <https://console.aws.amazon.com/mediatailor/>.
2. In the navigation pane, choose **Channel assembly** > **Channels**.
3. On the navigation bar, choose **Create channel**.
4. Under **Channel details**, enter details about your channel:
 - **Name:** Enter a name for your channel.
 - **Playback mode:** Determines what kind of program transitions are allowed and what happens to a program after it finishes. Use the default loop mode.
5. Choose **Next**.
6. Under **Output details**, define the settings for this output:
 - **Manifest name:** Enter a manifest name, such as *index*. MediaTailor will append the format extension, such as .m3u8 for HLS.

Note

You must enter a unique manifest name per channel output.

- **Format type:** Select the streaming format for the channel. DASH and HLS are supported. Choose the format that corresponds to the package configuration that you created in [Step 1: Create a source location](#).
 - **Source group:** Enter the name of the source group that you created in [Step 1: Create a source location](#).
7. Under **Manifest settings**, enter additional information about your manifest settings:
 - **Manifest window (sec):** The time window (in seconds) contained in each manifest. The minimum value is 30 seconds, and the maximum value is 3600 seconds.
 8. Choose **Next**.
 9. Under **Channel policy**, select **Do not attach channel policy**. This option restricts playback to only those who have access to your AWS account credentials.
 10. Choose **Next**.
 11. Review your settings on the **Review and create** pane.
 12. Choose **Create channel**.

Note

Channels are created in a stopped state. Your channel won't be active until you start it.

Step 4: Add programs to your channel's schedule

Now that you have a channel, you'll add *programs* to the channel's schedule. Each program contains a VOD source from a source location in your account. The channel schedule determines the order that your programs will play in the channel's stream.

Each program can have one or more ad breaks. You insert an ad break, by specifying a VOD source to use as an ad slate. The duration of the ad break is determined by the duration of the slate. You can optionally use a server-side ad insertion server, such as MediaTailor ad insertion, to personalize your ad breaks.

To add programs to your channel's schedule

1. Open the MediaTailor console at <https://console.aws.amazon.com/mediatailor/>.
2. In the navigation pane, choose **Channel assembly** > **Channels**.
3. In the **Channels** pane, choose the channel that you created in the [Step 3: Create a channel](#) procedure.
4. Under **Program details**, enter details about your program:
 - **Name:** This is the name of the program to add to your channel's schedule.
 - **Source location name:** Choose **Select an existing source location**, and select the source location that you created in the [Step 1: Create a source location](#) from the **Select a source location** drop-down menu.
 - **VOD source name:** Choose **Select an existing VOD source** and select the VOD source that you created earlier in this tutorial.
5. Under **Playback configuration**, define how and when a program is inserted in a channel's schedule:
 - **Transition type:** This value is fixed at **Relative**. The relative transition type indicates that this program occurs relative to other programs within the program list.
 - **Relative position:** If this is the first program in your channel's schedule, you can skip this setting. If it's not the first program in your channel's schedule, then choose where in the program list to append the program. You can select **Before program** or **After program**.
 - **Relative program:** If this is the first program in your schedule, you can skip this setting. If it's not the first program in your channel's schedule, then choose **Use existing program**, select the program name that you created in [To add programs to your channel's schedule](#).
6. Select **Add ad break**. Under **Ad breaks**, configure the settings for the ad break:
 - **Slate source location name:** Choose **Select an existing source location** and choose the source location where your slate is stored that you created earlier in this tutorial.
 - **VOD source name:** Choose **Select an existing VOD source** and choose the VOD source you're using for slate that you added earlier in this tutorial. The duration of the slate determines the duration of the ad break.
 - For **Offset in milliseconds:** This value determines the ad break start time in milliseconds, as an offset relative to the beginning of the program. Enter any value that's less than the duration of the VOD source, and that aligns with a segment boundary on all tracks within

the program's VOD source (all audio, video and closed caption tracks), otherwise the ad break will be skipped. For example, if you enter **0**, this creates a pre-roll ad break that plays before the program begins. Note: .

7. Choose **Add program**.

For more information about programs, see [Configuring ad breaks for your program](#).

For more advanced information about using ads with your linear stream, see [Optional configuration settings](#).

Step 5 (*optional*): Use MediaTailor to insert personalized ads into your stream

You now have a channel with programs. If you want, you can use MediaTailor to insert personalized ads into the ad breaks in your programs in the channel's stream.

Prerequisites

Before you proceed, you must meet the following requirements:

- You must have an ad decision server (ADS).
- You must have configured **Ad break** settings in the [Adding a program to a channel's schedule](#) procedure.

To add personalized ads to your channel's stream using MediaTailor

1. Open the MediaTailor console at <https://console.aws.amazon.com/mediatailor/>.
2. In the navigation pane, choose **Configurations**.
3. Under **Required settings**, enter the basic required information about your configuration:
 - **Name**: The name of your configuration.
 - **Content source**: Enter the playback URL from your channel's output, minus the file name and extension. For advanced information about MediaTailor configuration, see [Required settings](#).
 - **Add decision server**: Enter the URL for your ADS.
4. You can optionally configure the **Configuration aliases**, **Personalization details**, and **Advanced settings**. For information about those settings, see [Optional configuration settings](#).

5. On the navigation bar, choose **Create configuration**.

For more advanced information about using MediaTailor ad insertion, see [Using AWS Elemental MediaTailor to insert ads](#).

Step 6: Start your channel

You now have a channel. But before you can access the channel's stream, you need to start your channel. If you try to access a channel before it is active, MediaTailor returns an HTTP 4xx error code.

Start your channel

1. Open the MediaTailor console at <https://console.aws.amazon.com/mediatailor/>.
2. In the navigation pane, choose **Channel assembly** > **Channels**.
3. On the navigation bar, choose **Start**.

Step 7: Test your channel

To verify that your channel is working correctly, open a web browser and enter the URL from your channel's output. You should see your channel's stream.

In some cases, you might need to clear the cache to see the expected behavior.

Step 8: Clean up

After you've finished with the channel that you created for this tutorial, you should clean up by deleting the channel.

You'll stop incurring charges for that channel as soon as the channel status changes to stopped. To keep your channel for later, but not incur charges, you can stop the channel now and then start it again later.

To delete your channel

1. Open the MediaTailor console at <https://console.aws.amazon.com/mediatailor/>.
2. In the navigation pane, choose **Channel assembly** > **Channels**.
3. Select the channel that you want to delete.

4. If your channel is running, from the **Actions** drop-down menu, choose **Stop**. You must stop your channel before you can delete it.
5. When your channel is stopped, from the **Actions** drop-down menu, choose **Delete**.

Using AWS Elemental MediaTailor to insert ads

A configuration is an object that you interact with in AWS Elemental MediaTailor. The configuration holds the mapping information for the origin server and the ad decision server (ADS). You can also define a default playback for MediaTailor to use when an ad isn't available or doesn't fill the entire ad avail.

If you use a content delivery network (CDN) with MediaTailor, you must set up the behavior rules in the CDN before you add CDN information to the configuration. For more information about setting up your CDN, see [Using a CDN to optimize MediaTailor ad personalization and content delivery](#).

Topics

- [Supported audio and video codecs](#)
- [Understanding AWS Elemental MediaTailor ad insertion behavior](#)
- [MediaTailor server-guided ad insertion overview and implementation](#)
- [MediaTailor ad server integration requirements](#)
- [MediaTailor playback configuration management](#)
- [Integrating a content source for MediaTailor ad insertion](#)
- [Integrating AWS Elemental MediaTailor with Google's ad platforms](#)
- [Customizing ad break behavior with ad break suppression](#)
- [MediaTailor bumper ad insertion](#)
- [MediaTailor pre-roll ad insertion](#)
- [MediaTailor slate ad insertion](#)
- [Ad transcoding in AWS Elemental MediaTailor](#)
- [Prefetching ads](#)
- [Using preconditioned ads with AWS Elemental MediaTailor](#)
- [Working with MediaTailor Monetization Functions](#)
- [MediaTailor dynamic ad variables for ADS requests](#)
- [MediaTailor manifest query parameters](#)
- [Reporting ad tracking data](#)
- [Overlay ads](#)
- [Ad ID decoration](#)

Supported audio and video codecs

MediaTailor supports the following codecs.

- Audio codecs: mp4a, ac-3, and ec-3
- Video codecs: h.264 (AVC), h.265 (HEVC), av01 (AV1)

Understanding AWS Elemental MediaTailor ad insertion behavior

AWS Elemental MediaTailor stitches ads into live or video on demand (VOD) content by either replacing or inserting ads into the origin manifest. Whether ads are inserted or replaced depends on how the ad breaks are configured in the origin manifest, and whether the content is VOD or live. An ad break is the time period during programming when commercials are shown, while ad avails are the specific units of advertising time within an ad break that can be filled with ads.

- With *ad replacement*, MediaTailor replaces content segments with ads.
- With *ad insertion*, MediaTailor inserts ad content where segments don't exist.

For information about how MediaTailor stitches ads into live and VOD content, select the applicable topic.

Topics

- [Ad stitching behavior for VOD](#)
- [Live ad stitching behavior](#)

Ad stitching behavior for VOD

MediaTailor inserts or replaces ads in VOD content based on how ad markers are configured in the origin manifest, and if the ad decision server (ADS) sends VMAP responses.

For ad behavior by marker configuration, see the following sections.

If ad markers are present

AWS Elemental MediaTailor inserts ads where SCTE-35 ad markers are present in the origin manifest. Ad markers with an EXT-X-CUE-OUT value of 0 duration indicate ad insertion.

HLS ad marker guidelines

Follow these guidelines for post-roll and ad pod SCTE signaling:

Pre-roll ads

For HLS post-rolls, CUE-OUT/IN markers must precede the last content segment. This is because the HLS spec requires tag decorators to be explicitly declared before a segment.

For example, consider the following declaration.

```
#EXT-X-CUE-OUT: 0
#EXT-X-CUE-IN
#EXTINF:4.000,
Videocontent.ts
#EXT-X-ENDLIST
```

AWS Elemental MediaTailor inserts a post-roll like the following.

```
#EXTINF:4.000,
Videocontent.ts
#EXT-X-DISCONTINUITY
#EXTINF:3.0,
Adsegment1.ts
#EXTINF:3.0,
Adsegment2.ts
#EXTINF:1.0,
Adsegment3.ts
#EXT-X-ENDLIST
```

Example 2: Ad pods

CUE-OUT/IN tags must be explicitly attached to a segment. You can't use multiple CUE-OUT/IN tags in succession to mimic ad pod behavior.

For example, the following declaration is a valid use of CUE-OUT/IN to portray an ad pod.

```
#EXT-X-CUE-OUT: 0
```



```
#EXT-X-CUE-IN
#EXTINF:4.000,
Somecontent1.ts
#EXT-X-CUE-OUT: 0
#EXT-X-CUE-IN
#EXTINF:4.000,
Somecontent2.ts
#EXT-X-CUE-OUT: 0
#EXT-X-CUE-IN
#EXTINF:4.000,
Videocontent.ts
```

The preceding declaration results in an output like the following.

```
Ad 1
Somecontent.ts
Ad 2
Somecontent2.ts
Videocontent.ts
Post-Roll Ad 3
```

The following declaration is invalid.

```
#EXT-X-CUE-OUT: 0
#EXT-X-CUE-IN
#EXT-X-CUE-OUT: 0
#EXT-X-CUE-IN
#EXT-X-CUE-OUT: 0
#EXT-X-CUE-IN
#EXTINF:4.000,
Videocontent.ts
```

If no ad markers are present

Ad markers are the recommended way of signaling ad breaks in a manifest. However, ad markers aren't required. If the manifest doesn't contain ad markers for DASH or HLS, MediaTailor makes a single call to the ADS and creates ad breaks based on the response:

- If the ADS sends a VAST response, then MediaTailor inserts all ads from the response in an ad break at the start of the manifest. This is a pre-roll.

- If the ADS sends a VMAP response, then MediaTailor uses the ad break time offsets to create breaks and insert them throughout the manifest at the specified times (pre-roll, mid-roll, or post-roll). MediaTailor uses all ads from each ad break in the VMAP response for each ad break in the manifest.

Note

When a segment overlaps an insertion point with VMAP for VOD content, MediaTailor rounds down to the nearest insertion point.

Tip

If you want to create mid-roll ad breaks but your ADS doesn't support VMAP, then ensure that there are ad markers in the manifest. MediaTailor inserts ads at the markers, as described in the following sections.

Note

For server-guided ad insertion methods, MediaTailor inserts pre-roll ads at the top of the manifest and the player plays them before other ad types.

Live ad stitching behavior

In live streams, AWS Elemental MediaTailor always performs ad replacement, preserving the total time between the ad markers as closely as possible. When ad markers include the DURATION attribute, MediaTailor uses the value to determine the duration of the ad break. Every CUE-OUT indicator must have a duration or a matching CUE-IN indicator in live workflows.

MediaTailor performs ad replacement for HLS and DASH live content. For information on how MediaTailor calculates ad break placement and timing, see [the section called “Ad markers”](#) and [the section called “Ad markers”](#).

Ad selection and replacement

AWS Elemental MediaTailor includes ads from the ad decision server (ADS) VAST response as follows:

- If a duration is specified, MediaTailor selects a set of ads that fit into the duration and includes them.
- If no duration is specified, MediaTailor plays as many ads as it can until it encounters an ad marker that indicates a return to the main content.

AWS Elemental MediaTailor adheres to the following guidelines during live ad replacement:

- MediaTailor tries to play complete ads, without clipping or truncation.
- Whenever MediaTailor encounters an ad marker that indicates an end to the ad break, it returns to the underlying content. This can mean shortening an ad that is currently playing.
- At the end of the duration, MediaTailor returns to the underlying content.
- If MediaTailor runs out of ads to play for the duration of an ad break it either plays the slate, if one is configured, or resumes playback of the underlying content stream. This usually happens when there aren't enough transcoded ads to fill up the duration of the ad break.

Tip

You can define the limit of unfilled ad time allowed in an break with the personalization threshold configuration setting. For more information, see the [PlaybackConfiguration reference](#).

Live preroll for server-guided ad insertion

Live preroll works differently for server-guided ad insertion methods compared to server-side ad insertion:

Server-side ad insertion (stitched mode)

Preroll ads replace part of the live content at the beginning of each viewer's session. Each viewer sees preroll at different times based on when they join the stream.

Server-guided ad insertion methods

MediaTailor places a preroll datarange tag at the top of all media manifests with CUE="PRE , ONCE" attributes. This causes players to request and play preroll ads once at the start of playback, despite sharing the same non-personalized manifest.

Configuration requirements:

- **Live preroll ad decision server:** Configure a VAST endpoint for preroll ads (can be different from mid-roll ads)
- **Live preroll maximum allowed duration:** Set the maximum duration for preroll ads (optional - if omitted, all returned ads will be used)

Technical implementation:

- Preroll datarange tag uses START-DATE="1970-01-01T00:00:00.000Z" (Unix epoch)
- Asset list requests for preroll use the configured preroll ad decision server instead of the regular ADS
- Players identify preroll requests through the availId="aws-mediatailor-preroll-1" in the asset list data

Important

For live streams, preroll ads cover content rather than delaying it. Future versions may support content delay mode through additional configuration options.

Note

Preroll behavior varies between live and VOD content for server-guided ad insertion. Live content requires explicit preroll configuration, while VOD content includes preroll by default using the regular ad decision server.

Examples

- If the ad break has a duration set to 70 seconds and the ADS response contains two 40-second ads, AWS Elemental MediaTailor plays one of the 40-second ads. In the time left over, it switches to the configured slate or underlying content. At any point during this process, if MediaTailor encounters a cue-in indicator, it cuts immediately to the underlying content.
- If the ad break has a duration set to 30 seconds and the shortest ad provided by the ADS response is 40 seconds, MediaTailor plays no ads. If an ad slate is configured, MediaTailor plays that for 30 seconds or until it encounters a cue-in indicator. Otherwise, MediaTailor plays the underlying content.

MediaTailor server-guided ad insertion overview and implementation

AWS Elemental MediaTailor server-guided ad insertion (SGAI) provides an alternative to server-side ad insertion by referencing ads as separate playlists rather than stitching them directly into media playlists. This approach improves performance through cacheable manifests and enables better scalability.

For information about how to use server-guided ad insertion with MediaTailor, choose the applicable topic from the following list.

Enable in the playback configuration

In order to allow players to use server-guided ad insertion, you must set `Insertion Mode` to `PLAYER_SELECT` in the MediaTailor playback configuration. This allows players to select either stitched or guided ad insertion at session-initialization time.

Create a server-guided session

When creating playback sessions, choose guided mode. The way to do this depends on whether your players use implicit or explicit sessions.

Implicitly created server-guided sessions

Append `aws.insertionMode=GUIDED` to the HLS multivariant playlist request. Example:

```
playback-endpoint/v1/master/hashed-account-id/origin-id/index.m3u8?  
aws.insertionMode=GUIDED
```

Where:

- `playback-endpoint` is the unique playback endpoint that AWS Elemental MediaTailor generated when the configuration was created.

Example

```
https://777788889999.mediatailor.us-east-1.amazonaws.com
```

- `hashed-account-id` is your AWS account ID.

Example

```
777788889999
```

- `origin-id` is the name that you gave when creating the configuration.

Example

```
myOrigin
```

- `index.m3u8` or is the name of the manifest from the test stream plus its file extension. Define this so that you get a fully identified manifest when you append this to the video content source that you configured in [the section called “Step 4: Create a configuration”](#).

Using the values from the preceding examples, the full URLs are the following.

- Example:

```
https://777788889999.mediatailor.us-east-1.amazonaws.com/v1/master/777788889999/  
myOrigin/index.m3u8?aws.insertionMode=GUIDED
```

Explicitly created server-guided sessions

Add `insertionMode=GUIDED` to JSON metadata the player sends in the HTTP POST to the MediaTailor configuration's session-initialization prefix endpoint.

The following example shows the structure of the JSON metadata:

```
{
  # other keys, e.g. "adsParams"
  "insertionMode": "GUIDED"      # this can be either GUIDED or STITCHED
}
```

With this initialization metadata, the playback session will use server-guided ad insertion.

Ad tracking with SGAI

SGAI supports both server-side and client-side ad tracking. You set the reporting mode at session initialization. The mode cannot change during the session.

Server-side tracking (default)

MediaTailor fires VAST beacons automatically when the player requests ad segments. Ad URIs in the asset list contain encrypted beacon metadata (awsBeaconData, awsBeaconDomain, awsConfigurationName). The player must support HLS #EXT-X-DEFINE:QUERYPARAM variable substitution. The asset list response does not include a TRACKING section.

For details about how SGAI server-side beaconing works, see [Server-side tracking with server-guided ad insertion \(SGAI\)](#).

Client-side tracking

Add `aws.reportingMode=CLIENT` to your session initialization request. The asset list response includes a TRACKING section with beacon URLs that the player fires during ad playback. The GetTracking API endpoint is *not* used for SGAI sessions. Instead, each asset list response includes tracking data directly. The tracking data uses the same JSON schema as the server-side ad insertion (SSAI) tracking response.

For details, see [Server-guided ad insertion](#).

Guided prefetch with manifest heartbeating

For live SGAI streams, you can enable manifest-based ad prefetching by adding `aws.guidedPrefetchMode=MANIFEST` to your session initialization request:

```
https://777788889999.mediatailor.us-east-1.amazonaws.com/v1/master/777788889999/
myOrigin/index.m3u8?aws.insertionMode=GUIDED&aws.guidedPrefetchMode=MANIFEST
```

When enabled, MediaTailor appends a session identifier (`?aws.sessionId=<id>`) as a query parameter to each interstitial media manifest (`/v1/i-media`) URL in the multivariant playlist. Each time the player refreshes an i-media manifest, the request reaches MediaTailor with the session ID, which MediaTailor uses to enqueue prefetch requests for upcoming ad breaks.

- The `aws.guidedPrefetchMode` parameter accepts two values: MANIFEST (enabled) and OFF (disabled, default).
- Guided prefetch mode is only valid for SGAI sessions. Using it with stitched sessions returns an error.
- DASH does not yet support guided prefetch mode.
- Guided prefetch is independent of reporting mode — beacons fire at playback time, not at prefetch time.
- **Do not cache i-media manifests in your CDN when using guided prefetch.** The prefetch mechanism depends on the player's manifest refresh requests reaching MediaTailor directly. If your CDN caches `/v1/i-media` responses, MediaTailor does not receive the heartbeat requests and cannot trigger prefetching.

MediaTailor server-guided ad insertion feature compatibility matrix

AWS Elemental MediaTailor offers two ad insertion methods with different feature compatibility. Server-guided ad insertion works differently from server-side ad insertion, which affects compatibility with some MediaTailor features. Use this table to understand which features work with each ad insertion method.

Feature compatibility by ad insertion method

Feature	Server-side ad insertion (SSAI)	Server-guided ad insertion (SGAI)
Ad prefetching	✓ Supported	Not yet supported
Ad suppression	✓ Supported	Not applicable
Pre-roll ad behavior	Controlled by MediaTailor configuration	Controlled by MediaTailor configuration

Feature	Server-side ad insertion (SSAI)	Server-guided ad insertion (SGAI)
Client-side ad tracking	Uses GetTracking API	Uses TRACKING section in asset list (GetTracking API is not used)
Server-side ad tracking	✓ Supported — beacons fire based on /v1/segment requests using the session ID	✓ Supported (HLS only) — uses sessionless beaconing with encrypted beacon data embedded in ad URLs via #EXT-X-DEFINE:QUERYPARAM . Requires HLS v11 or later. DASH is not yet supported.
Ad-ID decoration	✓ Supported	✗ Not compatible

Compatibility details

Ad prefetching

Ad prefetching isn't currently supported.

Ad suppression

Ad suppression isn't supported with server-guided ad insertion methods because players only fetch ads they're going to play.

Pre-roll ad behavior

Pre-roll ad timing works differently between insertion methods:

- **Server-side ad insertion:** MediaTailor controls when pre-roll ads play based on configuration settings
- **Server-guided ad insertion:** MediaTailor inserts pre-roll ads at the top of the manifest. Your player shows these ads first, then starts your content

Ad tracking

Client-side tracking uses different mechanisms depending on the ad insertion method:

- **Server-side ad insertion (SSAI):** Uses the `GetTracking` API endpoint
- **Server-guided ad insertion (SGAI):** MediaTailor provides tracking information in the `TRACKING` section of each asset list response. The `GetTracking` API endpoint is not used. The session initialization response does not include a `trackingUrl`.

Server-side tracking also differs between methods:

- **Server-side ad insertion (SSAI):** MediaTailor fires beacons when the player fetches stitched ad segments through `/v1/segment/` using the session ID.
- **Server-guided ad insertion (SGAI):** MediaTailor uses sessionless beaconing. MediaTailor embeds encrypted beacon data (`awsBeaconData`, `awsBeaconDomain`, `awsConfigurationName`) in the ad manifest URLs that it returns in the asset list. The ad manifest uses `#EXT-X-DEFINE:QUERYPARAM` tags so the player substitutes these values into segment URLs. When the player requests each ad segment, MediaTailor decrypts the data, fires the appropriate beacon, and redirects to the content segment. When server-side reporting is active, MediaTailor omits the `TRACKING` section from the asset list response. For details, see [Server-side tracking with server-guided ad insertion \(SGAI\)](#).

Ad-ID decoration

Ad-ID decoration is not compatible with server-guided ad insertion because the fields that populate `X-AD-CREATIVE-SIGNALING` headers are only known when the asset list is fetched, not when the manifest is written.

MediaTailor server-guided ad insertion configuration for live streams

AWS Elemental MediaTailor server-guided ad insertion for live content provides significant performance benefits through cacheable manifests. Configuring SGAI for live content uses the

same core parameters as VOD, with specific considerations for live stream characteristics and real-time processing.

Requirements for live SGAI

Before you enable SGAI for live content, ensure that you have the following:

- Your live stream includes properly formatted DATERANGE markers
- Ad break durations are consistent and predictable
- Your CDN is configured to cache SGAI manifests appropriately
- Players support server-guided ad insertion workflows
- Your ad decision server can handle real-time requests for live content

Player requirements

Players must be configured to handle SGAI live manifests properly:

- Support for server-guided ad insertion workflows
- Ability to process ad insertion guidance from manifests
- Proper handling of live stream timing and synchronization
- For HLS content: Support for HLS version 8 and EXT-X-DATERANGE with CLASS attribute. Version 11 for server-side beaconing.
- For HLS content: EXT-X-DEFINE variable substitution support

Live playback configuration

To enable SGAI for live content, create a playback configuration that has the following settings:

Example SGAI live playback configuration

```
{
  "Name": "LiveSGAIConfig",
  "VideoContentSourceUrl": "https://your-live-origin.com/live/stream.m3u8",
  "AdDecisionServerUrl": "https://your-ads.com/ads",
  "PersonalizationThresholdSeconds": 1,
  "InsertionMode": "PLAYER_SELECT"
```

```
}
```

The following are key considerations for live SGAI configuration:

VideoContentSourceUrl

Must point to a live HLS stream with properly formatted SCTE-35 DATERANGE markers. The stream should maintain consistent segment durations and bitrate variants.

SGAI live manifest requests

SGAI live manifests use the same URL pattern as traditional ad insertion:

```
https://your-config.mediatailor.region.amazonaws.com/v1/master/config-name/  
manifest.m3u8?aws.insertionMode=GUIDED
```

Manifest-based prefetch for live SGAI

For live SGAI workflows, you can enable manifest-based prefetch heartbeating to reduce ad fill latency. Add `aws.guidedPrefetchMode=MANIFEST` to the manifest request:

```
https://your-config.mediatailor.region.amazonaws.com/v1/master/config-name/  
manifest.m3u8?aws.insertionMode=GUIDED&aws.guidedPrefetchMode=MANIFEST
```

When enabled, MediaTailor appends a session identifier (`?aws.sessionId=<id>`) as a query parameter to each interstitial media manifest (`/v1/i-media`) URL in the multivariant playlist. Each time the player refreshes an i-media manifest, the request reaches MediaTailor with the session ID, which MediaTailor uses to identify the session and enqueue prefetch requests for upcoming ad breaks.

Important

Do not cache i-media manifests in your CDN when using guided prefetch. The prefetch heartbeating mechanism depends on the player's manifest refresh requests reaching MediaTailor directly. If your CDN caches and serves `/v1/i-media` responses, MediaTailor does not receive the heartbeat requests and cannot trigger prefetching.

Configure your CDN to pass through `/v1/i-media/*` requests to MediaTailor when `aws.guidedPrefetchMode=MANIFEST` is in use.

Guided prefetch is independent of the reporting mode. Whether you use server-side (default) or client-side (`aws.reportingMode=CLIENT`) tracking, beacons fire at playback time, not when ads are prefetched. For general information about how ad prefetching works in MediaTailor, see [Prefetching ads](#).

Testing SGAI live configuration

Verify your SGAI live setup with these validation steps:

1. Test manifest generation

Request the SGAI live manifest URL and verify it returns cacheable content with proper ad insertion guidance.

2. Verify CDN caching

Check that your CDN is caching SGAI manifests according to the configured TTL values.

3. Test ad insertion

Confirm that players can successfully insert ads using the guidance provided in SGAI manifests.

4. Monitor performance

Use CloudWatch metrics to verify reduced origin load and improved cache hit rates.

MediaTailor server-guided ad insertion configuration for VOD content

AWS Elemental MediaTailor server-guided ad insertion for VOD content provides significant performance benefits through highly cacheable manifests and reduced server processing. Configuring SGAI for VOD content leverages the static nature of video-on-demand assets to maximize caching efficiency and minimize origin requests, making it ideal for large content libraries with repeated viewing patterns.

Requirements for VOD SGAI

Before you enable SGAI for VOD content, ensure that you have the following:

- Your VOD content includes properly formatted ad markers (SCTE-35 or timed metadata)
- Content is stored in a reliable origin with consistent availability
- Your CDN is configured to cache SGAI manifests with appropriate TTL values
- Players support server-guided ad insertion workflows
- Your ad decision server can handle VOD-specific metadata and targeting

Player requirements

Players must be configured to handle SGAI VOD manifests and ad insertion:

- Support for server-guided ad insertion workflows
- Ability to process ad insertion guidance from VOD manifests
- Support for client-side ad insertion during VOD playback
- Proper handling of seek operations across ad breaks
- Support for content duration and position tracking

VOD playback configuration

To enable SGAI for VOD content, create a playback configuration that has the following settings:

Example SGAI VOD playback configuration

```
{
  "Name": "VODSGAIConfig",
  "VideoContentSourceUrl": "https://your-vod-origin.com/content/",
  "AdDecisionServerUrl": "https://your-ads.com/ads",
  "PersonalizationThresholdSeconds": 5,
  "InsertionMode": "PLAYER_SELECT"
}
```

The following are key considerations for VOD SGAI configuration:

VideoContentSourceUrl

Should point to your VOD content library with consistent URL patterns. Ensure the origin can handle the expected request volume and provides reliable content delivery.

ConfigurationAliases

Include VOD-specific parameters like content duration, genre, or series information that can be used for ad targeting without affecting manifest cacheability.

ManifestProcessingRules

Enable ad marker passthrough to preserve original content timing information, which is especially important for VOD content with pre-defined ad break positions.

SGAI VOD manifest requests

SGAI VOD manifests use the same URL pattern traditional VOD ad insertion.

```
https://your-config.mediatailor.region.amazonaws.com/v1/master/config-name/content-path/manifest.m3u8?aws.insertionMode=GUIDED
```

VOD-specific ad targeting

VOD content enables unique ad targeting opportunities:

Content metadata targeting

Leverage VOD content metadata for improved ad targeting:

- **Genre and category:** Target ads based on content type (drama, comedy, documentary)
- **Content rating:** Ensure age-appropriate ad content (G, PG, R ratings)
- **Series and season:** Target ads for series continuity or related content
- **Release date:** Target based on content age (new releases vs. catalog content)
- **Content duration:** Adjust ad load based on total content length

Viewing context targeting

VOD viewing patterns enable contextual ad targeting:

- **Time of day:** Target ads based on when content is being watched

- **Binge watching:** Adjust ad frequency for users watching multiple episodes
- **Completion rate:** Target based on user's historical content completion patterns
- **Device type:** Optimize ad formats for viewing device (TV, mobile, tablet)

Testing SGAI VOD configuration

Verify your SGAI VOD setup with these validation steps:

1. Test manifest generation

Request SGAI VOD manifest URLs for different content types and verify they return cacheable content with proper ad insertion guidance.

2. Verify CDN caching

Check that your CDN is caching SGAI manifests according to the configured TTL values and achieving high cache hit rates.

3. Test ad insertion

Confirm that players can successfully insert ads using the guidance provided in SGAI manifests for various VOD content.

4. Test seek operations

Verify that seeking within VOD content works correctly across ad breaks and maintains proper playback position.

5. Monitor performance

Use CloudWatch metrics to verify reduced origin load, improved cache hit rates, and successful ad insertion rates.

Key testing scenarios

Test these specific VOD scenarios:

- **Popular content:** Verify high cache hit rates for frequently accessed VOD assets
- **Long-form content:** Test ad insertion in movies or long episodes with multiple ad breaks
- **Series content:** Verify consistent ad targeting across episodes in a series
- **Different genres:** Test ad targeting based on content metadata and genre

VOD SGAI optimization best practices

Optimize your SGAI VOD implementation for maximum performance:

Cache optimization

- **Maximize TTL values:** Use longer cache durations for VOD manifests since content doesn't change
- **Minimize cache keys:** Reduce cache key variations to improve hit rates
- **Pre-warm popular content:** Cache manifests for trending or featured VOD content
- **Monitor cache performance:** Track cache hit rates and optimize based on usage patterns

Content delivery optimization

- **Consistent URL patterns:** Use predictable URL structures for better caching
- **Metadata standardization:** Ensure consistent content metadata for reliable ad targeting
- **Ad break positioning:** Optimize ad break placement for natural content transitions
- **Quality variants:** Ensure SGAI works across all bitrate variants of your VOD content

MediaTailor ad server integration requirements

To integrate your ad server with AWS Elemental MediaTailor, your ad server must send XML that conforms to the IAB specifications for the supported versions of VAST and VMAP. You can use a public VAST validator to ensure that your tags are well-formed.

AWS Elemental MediaTailor supports VAST and VMAP responses from ad decision servers. AWS Elemental MediaTailor also supports the proxying of VPAID metadata through our client-side reporting API for client-side ad insertion. For information about client-side reporting, see [Client-side ad tracking](#).

MediaTailor supports the following versions of VAST, VMAP, and VPAID:

- Up to [VAST 4.3](#)

MediaTailor accepts response versions through VAST 4.3, but some advanced features from VAST 4.0 and up are not supported.

- [VMAP 1.0](#)

- [VPAID 2.0](#)

VAST requirements

Your ad server's VAST response must contain IAB compliant `TrackingEvents` elements and standard event types, like `impression`. If you include non-standard tracking events, AWS Elemental MediaTailor rejects the VAST response and doesn't provide an ad for the avail.

VAST 3.0 introduced support for ad pods, which is the delivery of a set of sequential linear ads. If a specific ad in an ad pod is not available, AWS Elemental MediaTailor logs an error on CloudWatch, in the interactions log of the ADS. It then tries to insert the next ad in the pod. In this way, MediaTailor iterates through the ads in the pod until it finds one that it can use.

Targeting

To target specific players for your ads, you can create templates for your ad tags and URLs. For more information, see [MediaTailor dynamic ad variables for ADS requests](#).

AWS Elemental MediaTailor proxies the player's `user-agent` and `x-forwarded-for` headers when it sends the ad server VAST request and when it makes the server-side tracking calls. Make sure that your ad server can handle these headers. Alternatively, you can use `[session.user_agent]` or `[session.client_ip]` and pass these values in query strings on the ad tag and ad URL. For more information, see [MediaTailor session variables for ADS requests](#).

Ad calls

AWS Elemental MediaTailor calls your VAST ads URL as defined in your configuration. It substitutes any player-specific or session-specific parameters when making the ad call. MediaTailor follows up to seven levels of VAST wrappers and redirects in the VAST response. In live streaming scenarios, MediaTailor makes ad calls simultaneously at the ad avail start for connected players. In practice, due to jitter, these ad calls can be spread out over a few seconds. Make sure that your ad server can handle the number of concurrent connections this type of calling requires. MediaTailor supports prefetching VAST responses for live workflows. For more information, see [Prefetching ads](#).

Creative handling

When AWS Elemental MediaTailor receives the ADS VAST response, for each creative it identifies the highest bitrate `MediaFile` for transcoding and uses this as its source. It sends this file to the

on-the-fly transcoder for transformation into renditions that fit the player's multivariant playlist bitrates and resolutions. For best results, make sure that your highest bitrate media file is a high-quality MP4 asset with valid manifest presets. When manifest presets aren't valid, the transcode jobs fail, resulting in no ad shown. Examples of presets that aren't valid include unsupported input file formats, like ProRes, and certain rendition specifications, like the resolution 855X481.

For a list of supported formats for media file inputs, see the **MP4** row of [Supported input formats](#) in the *AWS Elemental MediaConvert User Guide*.

Creative indexing

AWS Elemental MediaTailor uniquely indexes each creative by the value of the `id` attribute provided in the `<Creative>` element. If a creative's ID is not specified, MediaTailor uses the media file URL for the index.

The following example declaration shows the creative ID.

```
<Creatives>
  <Creative id="57859154776" sequence="1">
```

If you define your own creative IDs, use a new, unique ID for each creative. Don't reuse creative IDs. AWS Elemental MediaTailor stores creative content for repeated use, and finds each by its indexed ID. When a new creative comes in, the service first checks its ID against the index. If the ID is present, MediaTailor uses the stored content, rather than reprocessing the incoming content. If you reuse a creative ID, MediaTailor uses the older, stored ad and doesn't play your new ad.

VAST extensions provided by ad-serving partners

To help prevent collisions with creative IDs, you can use extensions provided by ad-serving partners for the VAST response. MediaTailor supports extensions from SpringServe, Publica, and FreeWheel. When you enable VAST extension overrides, MediaTailor replaces the default creative ID with the extension value.

To enable this feature, [submit an AWS Support ticket](#) to request VAST extension-based creative IDs to be enabled. Include the following information in the Support ticket:

- AWS Region
- AWS account ID
- MediaTailor playback configuration names

To validate that VAST extension-based creative IDs are enabled on your account, we recommend that you also request `RAW_ADS_RESPONSE` logging to be enabled on a staging or testing playback configuration. With logging, you can view the original VAST response that the ADS receives and confirm that the correct creative IDs are used.

VPAID requirements

VPAID allows publishers to serve highly interactive video ads and to provide viewability metrics on their monetized streams. For information about VPAID, see the [VPAID specification](#).

AWS Elemental MediaTailor supports a mix of server-side-stitched VAST MP4 linear ads and client-side-inserted VPAID interactive creatives in the same ad avail. It preserves the order in which they appear in the VAST response. MediaTailor follows VPAID redirects through a maximum of seven levels of wrappers. The client-side reporting response includes the unwrapped VPAID metadata.

To use VPAID, follow these guidelines:

- Configure an MP4 slate for your VPAID creatives. AWS Elemental MediaTailor fills the VPAID ad slots with your configured slate, and provides VPAID ad metadata for the client player to use to run the interactive ads. If you don't have a slate configured, when a VPAID ad appears, MediaTailor provides the ad metadata through client-side reporting as usual. It also logs an error in CloudWatch about the missing slate. For more information, see [MediaTailor slate ad insertion](#) and [Creating an MediaTailor playback configuration](#).
- Use client-side reporting. AWS Elemental MediaTailor supports VPAID through our client-side reporting API. For more information, see [Client-side ad tracking](#).

It is theoretically possible to use the default server-side reporting mode with VPAID. However, if you use server-side reporting, you lose any information about the presence of the VPAID ad and the metadata surrounding it, because that is available only through the client-side API.

- In live scenarios, make sure that your ad avails, denoted by `EXT-X-CUE-OUT: Duration`, are long enough to accommodate any user interactivity on VPAID. For example, if the VAST XML specifies a VPAID ad that is 30 seconds long, consider configuring your ad avail to be more than 30 seconds. This additional time gives users more chance to interact with the ad. If you don't add time, you could lose the VPAID metadata because the remaining duration in the ad avail is not long enough to accommodate the VPAID ad.

MediaTailor playback configuration management

This section covers the key tasks for managing AWS Elemental MediaTailor playback configurations. You can learn how to create a new configuration to set up content streams and provide access for playback devices, view details of an existing configuration, edit a configuration to update settings like origin servers and ad decision servers, and delete a configuration that is no longer needed.

Topics

- [Creating an MediaTailor playback configuration](#)
- [Viewing MediaTailor configuration details](#)
- [Editing MediaTailor configuration settings](#)
- [Deleting MediaTailor configurations](#)

Creating an MediaTailor playback configuration

This topic shows how to create a configuration to start receiving content streams with AWS Elemental MediaTailor. It also shows how to provide an access point for downstream playback devices to request content.

You can use the AWS Elemental MediaTailor console, the AWS Command Line Interface (AWS CLI)>, or the MediaTailor API to create a configuration. For information about creating a configuration through the AWS CLI or MediaTailor API, see the [AWS Elemental MediaTailor API Reference](#)..

When you create a configuration, don't put sensitive identifying information into free-form fields such as the **Configuration name** field. Identifying information can include things like customer account numbers. In addition, don't use identifying information when you work in the MediaTailor console, REST API, the AWS CLI, or AWS SDKs. Any data that you enter into MediaTailor might get picked up for inclusion in diagnostic logs or Amazon CloudWatch Events.

To add a configuration (console)

1. Open the MediaTailor console at <https://console.aws.amazon.com/mediatailor/>.
2. On the **Configurations** page, choose **Create configuration**.
3. Complete the configuration and additional configuration fields as described in the following topics:

- [Required settings](#)
 - [Optional configuration settings](#)
4. Choose **Create configuration**.

AWS Elemental MediaTailor displays the new configuration in the table on the **Configurations** page.

5. (Recommended) Set up a CDN with AWS Elemental MediaTailor for manifest and reporting requests. You can use the configuration playback URLs for the CDN setup. For information about setting up a CDN for manifest and reporting requests, see [Using a CDN to optimize MediaTailor ad personalization and content delivery](#).

Required settings

When you create a configuration, you must include the following required settings.

Name

Enter a unique name that describes the configuration. The name is the primary identifier for the configuration. The maximum length allowed is 512 characters.

Content source

Enter the URL prefix for the manifest for this stream, minus the asset ID. The maximum length is 512 characters.

For example, the URL prefix `http://origin-server.com/a/` is valid for an HLS multivariant playlist URL of `http://origin-server.com/a/main.m3u8` and for a DASH MPD URL of `http://origin-server.com/a/dash.mpd`. Alternatively, you can enter a shorter prefix such as `http://origin-server.com`, but the `/a/` must be included in the asset ID in the player request for content.

Note

If your content origin uses HTTPS, its certificate must be from a well-known certificate authority. It can't be a self-signed certificate. If you use a self-signed certificate, AWS Elemental MediaTailor fails to connect to the content origin and can't serve manifests in response to player requests.

Ad decision server

Enter the URL for your ad decision server (ADS). This is either the URL with variables as described in [Step 3: Configure ADS request URL and query parameters](#) , or the static VAST URL that you are using for testing purposes. The maximum length is 25,000 characters.

Note

If your ADS uses HTTPS, its certificate must be from a well-known certificate authority. It can't be a self-signed certificate. The same also applies to mezzanine ad URLs returned by the ADS. If you use a self-signed certificate, then AWS Elemental MediaTailor can't retrieve and stitch ads into the manifests from the content origin.

Optional configuration settings

You can optionally configure **configuration aliases**, **personalization details**, and **advanced settings** in the MediaTailor console, MediaTailor API, or the AWS Command Line Interface (AWS CLI).

Configuration aliases

The following are optional configuration aliases that you can configure in the MediaTailor console, or with the MediaTailor API.

Player parameter variable

For dynamic domain configuration during session initialization, add one or more player parameter variables.

For more information about using player parameter variables to dynamically configure domains, see [MediaTailor domain variables for multiple content sources](#).

Log configuration

The following are log configuration settings.

Percent enabled

Sets the percentage of playback configuration session logs that MediaTailor writes to CloudWatch Logs. For example, if your playback configuration has 1000 sessions, and you set percent enabled to **60**, MediaTailor writes 600 session logs to CloudWatch Logs.

When you enable this option, MediaTailor automatically creates a service-linked role that allows MediaTailor to write and manage session logs in your CloudWatch Logs account. For more information, see [Using service-linked roles for MediaTailor](#).

Logging strategies

Indicates the method used for collecting logs that MediaTailor emits. To send logs directly to CloudWatch Logs, choose `LEGACY_CLOUDWATCH`. To send logs to CloudWatch Logs, which then vends logs to your destination of choice, choose `VENDED_LOGS`. Supported destinations are a CloudWatch Logs log group, Amazon S3 bucket, and Amazon Data Firehose stream.

Additional set up is required for vended logs. For set up, see [Using vended logs](#).

ADS interaction log opt-in events

Indicates that MediaTailor will emit `RAW_ADS_RESPONSE` logs for sessions that are initialized with this configuration.

The `RAW_ADS_RESPONSE` log event contains the entire VAST or VMAP response from the ADS. As such, the logs can be extensive and might increase your logging costs.

ADS interaction log exclude events

Indicates that MediaTailor won't emit the selected events in the logs that describe interactions with the ADS.

For a description of ADS log events, see [ADS logs](#).

Manifest service interaction log exclude events

Indicates that MediaTailor won't emit the selected events in the logs that describe interactions with the manifest service.

For a description of manifest service log events, see [Manifest logs](#).

Ad conditioning

The following determine what actions MediaTailor takes to condition ads before stitching them into a content stream.

Streaming media file conditioning

Determines the logic that MediaTailor uses when deciding which ads to stitch.

- When **Streaming media file conditioning** is set to **Transcode**, MediaTailor transcodes the media files with progressive delivery and stitches them into the manifest. If there aren't enough ads with progressive delivery media files to fill the avail, MediaTailor transcodes and uses those with streaming delivery.
- When **Streaming media file conditioning** is set to **None**, MediaTailor stitches ads with streaming delivery media files into the manifest without transcoding them. If there aren't enough ads with streaming delivery media files to fill the avail, MediaTailor transcodes and uses those with progressive delivery.

Personalization details

The following are personalization details that you can configure in the MediaTailor console or with the MediaTailor API.

Slate ad

Enter the URL for a high-quality MP4 asset that MediaTailor uses to fill unfilled ad time. Slate configuration is optional for most workflows but mandatory for VPAID ads. For comprehensive information about slate behavior, configuration requirements, and when slate is shown, see [MediaTailor slate ad insertion](#).

Start bumper

The URL of the start bumper asset location. Bumpers are short video or audio clips that play at the beginning or end of an ad break. They can be stored on Amazon's S3, or a different storage service. To learn more about bumpers, see [MediaTailor bumper ad insertion](#).

End bumper

The URL of the end bumper asset location. Bumpers are short video or audio clips that play at the beginning or end of an ad break. They can be stored on Amazon's S3, or a different storage service. To learn more about bumpers, see [MediaTailor bumper ad insertion](#).

Personalization threshold

Only applies when used in conjunction with slate ad. Defines the maximum duration of underfilled ad time (in seconds) allowed in an ad break before MediaTailor abandons personalization and shows underlying content. This feature applies to ad replacement in live and VOD streams, rather than ad insertion, because it relies on an underlying content stream. For detailed behavior scenarios and comprehensive information, see [How personalization threshold works](#).

Live pre-roll ad decision server

To insert ads at the start of a live stream before the main content starts playback, enter the URL for the ad pre-roll from the ad decision server (ADS). This is either the URL with variables as described in [Step 3: Configure ADS request URL and query parameters](#), or the static VAST URL that you are using for testing purposes. The maximum length is 25,000 characters.

Note

If your ADS uses HTTPS, its certificate must be from a well-known certificate authority. It can't be a self-signed certificate. The same also applies to mezzanine ad URLs returned by the ADS. If you use a self-signed certificate, then AWS Elemental MediaTailor can't retrieve and stitch ads into the manifests from the content origin.

For information about how pre-roll works, see [MediaTailor pre-roll ad insertion](#).

Live pre-roll maximum allowed duration

When you're inserting ads at the start of a live stream, enter the maximum allowed duration for the pre-roll ad avail. MediaTailor won't go over this duration when inserting ads. If the response from the ADS contains more ads than will fit in this duration, MediaTailor fills the avail with as many ads as possible, without going over the duration. For more details about how MediaTailor fills avails, see [Live ad stitching behavior](#).

Avail suppression mode

Sets the mode for avail suppression, also known as ad suppression. By default, ad suppression is off and MediaTailor fills all with ads or slate. When the mode is set to `BEHIND_LIVE_EDGE`, ad suppression is active and MediaTailor doesn't fill ad breaks on or behind the avail suppression value time in the manifest lookback window. When the mode is set to `AFTER_LIVE_EDGE`,

ad suppression is active. MediaTailor doesn't fill ad breaks on or behind the avail suppression period, which is the live edge plus the avail suppression value plus buffer time.

Avail suppression value

The avail suppression value is a live edge offset time in HH:MM:SS. MediaTailor won't fill ad breaks on or behind this time in the manifest lookback window.

Insertion Mode

Insertion Mode controls whether players can use stitched or guided ad insertion. The default, STITCHED_ONLY, forces all player sessions to use stitched (server-side) ad insertion. Setting InsertionMode to PLAYER_SELECT allows players to select either stitched or guided ad insertion at session-initialization time. The default for players that do not specify an insertion mode is stitched.

Advanced settings

The following are optional advanced settings. You can configure these in the MediaTailor console, with the AWS Command Line Interface (AWS CLI), or using the MediaTailor API.

CDN content segment prefix

Enables AWS Elemental MediaTailor to create manifests with URLs to your CDN path for content segments. Before you do this step, set up a rule in your CDN to pull segments from your origin server. For **CDN content segment prefix**, enter the CDN prefix path.

For more information about integrating MediaTailor with a CDN, see [Using a CDN to optimize MediaTailor ad personalization and content delivery](#).

CDN ad segment prefix

Enables AWS Elemental MediaTailor to create manifests with URLs to your own CDN path for ad segments. By default, MediaTailor serves ad segments from an internal Amazon CloudFront distribution with default cache settings. Before you can complete the **CDN ad segment prefix** field, you must set up a rule in your CDN to pull ad segments from the following origin, like in the following example.

```
https://segments.mediataylor.<region>.amazonaws.com
```

For **CDN ad segment prefix**, enter the name of your CDN prefix in the configuration.

For more information about integrating MediaTailor with a CDN, see [Using a CDN to optimize MediaTailor ad personalization and content delivery](#).

DASH origin manifest type

If your origin server produces single-period DASH manifests, open the dropdown list and choose **SINGLE_PERIOD**. By default, MediaTailor handles DASH manifests as multi-period manifests.

For more information, see [the section called “Integrating an MPEG-DASH source”](#).

DASH mpd location

(Optional as needed for DASH) The media presentation description (mpd) location. Choose **DISABLED** for the following situation:

- You set up CDN routing rules for accessing MediaTailor manifests.
- You use client-side reporting, or your player supports sticky HTTP redirects.

For more information about the **Location** feature, see [the section called “Location feature”](#).

Transcode profile name

The name that associates this configuration with a custom transcode profile. This name overrides the dynamic transcoding defaults of MediaTailor. Complete this field only if you have already set-up custom profiles with the help of AWS Support.

Ad marker passthrough

For HLS, enables or disables ad marker passthrough. When ad marker passthrough is enabled, MediaTailor passes through EXT-X-CUE-IN, EXT-X-CUE-OUT, and EXT-X-SPLICEPOINT-SCTE35 ad markers from the origin manifest to the MediaTailor personalized manifest. No logic is applied to the ad marker values; they are passed from the origin manifest to the personalized manifest as-is. For example, if EXT-X-CUE-OUT has a value of 60 in the origin manifest, but no ads are placed, MediaTailor won't change the value to 0 in the personalized manifest.

Individual ADS request timeout

The maximum time, in milliseconds, that MediaTailor waits for a single HTTP response when resolving an ad decision during live or VOD playback. Default: 3000. Valid range: 250–6000.

For more information about how MediaTailor handles ad stitching in live and VOD workflows, see [Live ad stitching behavior](#) and [Ad stitching behavior for VOD](#).

This setting corresponds to the `AdsRequestTimeoutMilliseconds` member of the [PutPlaybackConfiguration](#) `AdsPersonalizationTimeouts` parameter.

Maximum concurrent ADS requests

The number of simultaneous requests that MediaTailor can make to the ADS. Default: 1. Valid range: 1–100.

This setting corresponds to the `MaxConcurrentAdsRequests` member of the [PutPlaybackConfiguration](#) `AdsPersonalizationConcurrency` parameter.

Live maximum ADS personalization time

The total time budget, in milliseconds, for all ADS activity during live manifest personalization. This budget is shared across all ad breaks in a single manifest request. Default: 10000. Valid range: 250–10000.

For more information about live ad stitching behavior, see [Live ad stitching behavior](#).

This setting corresponds to the `LiveMaximumAdsPersonalizationTimeMilliseconds` member of the [PutPlaybackConfiguration](#) `AdsPersonalizationTimeouts` parameter.

VOD maximum ADS personalization time

The total time budget, in milliseconds, for all ADS activity during VOD manifest personalization. This budget is shared across all ad breaks in a single manifest request. Default: 10000. Valid range: 250–10000.

For more information about VOD ad stitching behavior, see [Ad stitching behavior for VOD](#).

This setting corresponds to the `VodMaximumAdsPersonalizationTimeMilliseconds` member of the [PutPlaybackConfiguration](#) `AdsPersonalizationTimeouts` parameter.

Enable VOD VAST parallelization

When you turn on this setting, MediaTailor processes ADS requests in parallel for VOD workflows that return VAST responses. This setting applies to VOD workflows only. MediaTailor turns off this setting by default. To take effect, set **Maximum concurrent ADS requests** to a value greater than 1.

Frequency capping with concurrent requests

When you enable VAST parallelization, some ad servers might not enforce frequency cap limits for concurrent ad requests. Check with your ad server provider to confirm whether frequency capping is supported under parallel request conditions. If your ad

server doesn't support frequency caps with concurrent requests, consider switching to VMAP, which handles frequency capping correctly for parallel ad requests.

This setting corresponds to the `EnableVodVastParallelization` member of the [PutPlaybackConfiguration](#) `AdsPersonalizationConcurrency` parameter.

Prefetch ADS request timeout

The maximum time, in milliseconds, that MediaTailor waits for a single HTTP response when resolving an ad decision during prefetch retrieval. Default: the **Individual ADS request timeout** value. Valid range: 250–10000.

For more information about prefetch, see [Prefetching ads](#).

This setting corresponds to the `PrefetchAdsRequestTimeoutMilliseconds` member of the [PutPlaybackConfiguration](#) `AdsPersonalizationTimeouts` parameter.

Prefetch maximum ADS personalization time

The total time budget, in milliseconds, for all ADS activity during prefetch retrieval. By default, this timeout has no upper limit. Valid range: 250–10000.

For more information about prefetch, see [Prefetching ads](#).

This setting corresponds to the `PrefetchMaximumAdsPersonalizationTimeMilliseconds` member of the [PutPlaybackConfiguration](#) `AdsPersonalizationTimeouts` parameter.

Setting timeouts together

For each workflow, you must set the individual timeout and the total time budget together. The individual timeout can't exceed the total budget. Live and VOD workflows share a single **Individual ADS request timeout**. Set that timeout together with both the live and VOD total time budgets. Similarly, set **Prefetch ADS request timeout** and **Prefetch maximum ADS personalization time** together — you can't set one without the other.

Viewing MediaTailor configuration details

In addition to the values provided when the configuration was created, AWS Elemental MediaTailor displays the name of the configuration, playback endpoints, and relevant access URLs. To view a configuration, use the following procedure.

To view a configuration

1. Open the MediaTailor console at <https://console.aws.amazon.com/mediatailor/>.
2. On the **Configurations** page, choose the **Configuration name** for the configuration to view.

Editing MediaTailor configuration settings

You can edit a configuration to update the origin server and ad decision server (ADS) mapping, or change how AWS Elemental MediaTailor interacts with a content distribution network (CDN).

To edit a configuration

1. Open the MediaTailor console at <https://console.aws.amazon.com/mediatailor/>.
2. On the **Configurations** page, choose the name of the configuration that you want to edit.
3. On the configuration details page, choose **Edit**, and then revise the configuration settings as needed. You can't edit the configuration name. For information about configuration attributes, see [Creating an MediaTailor playback configuration](#).
4. Choose **Save**.

Deleting MediaTailor configurations

You can delete a configuration to make it unavailable for playback with AWS Elemental MediaTailor.

To delete a configuration

1. Open the MediaTailor console at <https://console.aws.amazon.com/mediatailor/>.
2. On the **Configurations** page, do one of the following:
 - Choose the name of the configuration that you want to delete.

- In the **Configuration name** column, choose the option next to the name, and then choose **Delete**.
3. In the **Delete** confirmation box, enter **Delete**, and then choose **Delete**.

Integrating a content source for MediaTailor ad insertion

This topic covers integrating different types of video content sources with MediaTailor. MediaTailor supports both HLS and DASH streaming protocols for live and on-demand content. The service can perform ad insertion or replacement during designated ad breaks, and has specific requirements for the structure and formatting of the input video manifests to enable these capabilities. The following topics provide details on the input source requirements and steps for integrating HLS and DASH content with MediaTailor to enable personalized ad experiences.

Topics

- [Input source requirements for MediaTailor ad insertion](#)
- [Integrating an HLS source](#)
- [Integrating an MPEG-DASH source](#)
- [Securing AWS Elemental MediaTailor origin interactions with SigV4](#)

Input source requirements for MediaTailor ad insertion

A input source must meet the following requirements to work with MediaTailor:

- Use Apple HLS (HTTP Live Streaming) or MPEG DASH (Dynamic Adaptive Streaming over HTTP)
- Use live streaming or video on demand (VOD)
- Be accessible on the public internet and have a public IP address
- Use standard HTTP ports (port 80) or HTTPS ports (port 443). MediaTailor does not support custom ports for origin server communication.
- Contain ad markers in one of the formats described in the [Getting started with MediaTailor ad insertion tutorial](#)

Integrating an HLS source

AWS Elemental MediaTailor supports .m3u8 HLS manifests with an EXT-X-VERSION of 3 or higher for live streaming and video on demand (VOD). When MediaTailor encounters an ad break, it attempts ad insertion or replacement, based on the type of content. If there aren't enough ads to fill the duration, for the remainder of the ad break, MediaTailor displays the underlying content stream or the configured slate. For more information about HLS ad behavior based on content type, see [Understanding AWS Elemental MediaTailor ad insertion behavior](#).

The following sections provide more information about how MediaTailor handles HLS manifests.

Topics

- [HLS supported ad markers](#)
- [Enabling ad marker passthrough](#)
- [HLS manifest tag handling](#)
- [HLS manifest examples](#)

HLS supported ad markers

AWS Elemental MediaTailor identifies ad avail boundaries in an HLS manifest by parsing the input manifest for supported ad markers. The following sections describe what markers MediaTailor uses.

EXT-X-ASSET

The EXT-X-ASSET tag contains metadata that's used by the ad decision server (ADS) to personalize content for the viewer. EXT-X-ASSET parameters are comma-separated key-value pairs.

To use this tag, you must meet the following requirements:

- You must URL-encode the EXT-X-ASSET *values* in the origin manifest. The following example shows the EXT-X-ASSET tag with keys and URL-encoded values.

```
#EXT-X-ASSET:GENRE=CV,CAID=12345678,EPIISODE="Episode%20Name%20Date",SEASON="Season%20Name%20and%20Number",SERIES="Series%2520Name"
```

- You must include the dynamic `[asset.]` variable and the *keys* in your MediaTailor ADS configuration. The following example shows an MediaTailor ADS configuration using the dynamic `[asset.]` variable and keys.

```
https://myads.com/stub?  
c=[asset.GENRE]&g=[asset.CAID]&e=[asset.EPISODE]&s=[asset.SEASON]&k=[asset.SERIES]
```

Example VAST request

The following example shows a VAST GET request to an ADS.

```
https://myads.com/stub?c=CV&g=12345678&e=Episode%20Name%20Date&s=Season  
%20Name%20and%20Number&k=Series%2520Name
```

EXT-X-CUE-OUT and EXT-X-CUE-IN

This type of ad marker is the most common. The following examples show options for these cue markers.

```
#EXT-X-CUE-OUT:DURATION=120  
...  
#EXT-X-CUE-IN
```

```
#EXT-X-CUE-OUT:30.000  
...  
#EXT-X-CUE-IN
```

```
#EXT-X-CUE-OUT  
...  
#EXT-X-CUE-IN
```

EXT-X-DATERANGE

With EXT-X-DATERANGE ad marker tags, you use SCTE35-OUT attributes to specify the timing of the ad avail.

Note

AWS Elemental MediaTailor ignores any START-DATE attributes that are provided for EXT-X-DATERANGE ad markers.

You can specify the ad avail in one of the following ways:

- EXT-X-DATERANGE tag with SCTE35-OUT and DURATION specifications.

Example

```
#EXT-X-DATERANGE:ID="splice-6FFFFFFF0",START-DATE="2019-01T00:15:00Z\n",DURATION=60.000,SCTE35-OUT=0xF
```

- Paired EXT-X-DATERANGE tags, the first with a SCTE35-OUT specification and the second with a SCTE35-IN specification.

Example

```
#EXT-X-DATERANGE:ID="splice-6FFFFFFF0",START-DATE="2019-01T00:15:00Z\n",SCTE35-OUT=0xF
...
#EXT-X-DATERANGE:ID="splice-6FFFFFFF0",START-DATE="2019-01T00:15:00Z\n",SCTE35-IN=0xF
```

- A combination of the prior options. You specify an EXT-X-DATERANGE tag with SCTE35-OUT and DURATION specifications followed by an EXT-X-DATERANGE tag with a SCTE35-IN specification. In this case, MediaTailor uses the earliest cue-in setting from the two specifications.

Example

```
#EXT-X-DATERANGE:ID="splice-6FFFFFFF0",START-DATE="2019-01T00:15:00Z\n",DURATION=60.000,SCTE35-OUT=0xF
...
#EXT-X-DATERANGE:ID="splice-6FFFFFFF0",START-DATE="2019-01T00:15:00Z\n",SCTE35-IN=0xF
```

EXT-X-SPLICEPOINT-SCTE35

You append the EXT-X-SPLICEPOINT-SCTE35 ad marker tag with a SCTE-35 payload in base64-encoded binary. The decoded binary must provide a SCTE-35 `splice_info_section` containing the cue-out marker 0x34, for provider placement opportunity start, and the cue-in marker 0x35, for provider placement opportunity end.

The following example shows the splice point specification with base64-encoded binary payloads that specify the cue-out and cue-in markers.

```
#EXT-X-SPLICEPOINT-SCTE35:/DA9AAAAAAAAAAP/wBQb+uYbZqwAnAiVDVUVJAAAKqX//  
AAEjW4AMEU1EU05CMDAxMTMyMjE5M190NAAAmXz5JA==  
...  
#EXT-X-SPLICEPOINT-SCTE35:/DA4AAAAAAAAAAP/wBQb+tTeaawAiAiBDVUVJAAAKqH+/  
DBFNRFNOQjAwMTEzMjIxOTJfTjUAAIiGK1s=
```

Enabling ad marker passthrough

By default for HLS, MediaTailor personalized manifests don't include the SCTE-35 ad markers from the origin manifests. When ad marker passthrough is enabled, MediaTailor passes through the following ad markers from origin manifests into personalized manifests:

- EXT-X-CUE-IN
- EXT-X-CUE-OUT
- EXT-X-SPLICEPOINT-SCTE35

Ad marker passthrough is an optional setting. Use ad marker passthrough if you want the SCTE ad markers to be included in the MediaTailor personalized manifest. Common use cases include the following:

- Content replacement - Perform content replacement or content restriction.
- Ad tracking - Trigger ad tracking information based on the presence or absence of one or more ad markers.
- Player settings - Enable scrubbing or countdown timer functionality in the player's UI, based on the presence or absence of ad markers.

Note

MediaTailor doesn't change the values for these markers. For example, if EXT-X-CUE-OUT has a value of 60 in the origin manifest, but no ads are placed, MediaTailor won't change the value to 0 in the personalized manifest.

Enable ad marker passthrough

You can enable ad marker passthrough using the AWS Management Console or the AWS Command Line Interface (AWS CLI).

To enable ad marker passthrough using the console

1. Open the MediaTailor console at <https://console.aws.amazon.com/mediatailor/>.
2. Select either **New Configuration** or **Edit Configuration**.
3. In the **Advanced Settings** section, select **Enable** from the drop down menu.

To enable ad marker passthrough using the AWS Command Line Interface (AWS CLI)

Use the [put-playback-configuration](#) command.

HLS manifest tag handling

This section describes how AWS Elemental MediaTailor manages tags in the personalized output manifest.

EXT-X-CUE tags

MediaTailor replaces EXT-X-CUE-OUT, EXT-X-CUE-OUT-CONT, and EXT-X-CUE-IN tags in the input manifest with EXT-X-DISCONTINUITY tags in the output manifest. The DISCONTINUITY tags mark the following boundaries:

- Where the main content transitions to an ad
- Where one ad transitions to another ad
- Where an ad transitions back to the main content

EXT-X-DATERANGE tags

MediaTailor passes through EXT-X-DATERANGE tags from the input manifest to the output manifest. MediaTailor also inserts EXT-X-DISCONTINUITY tags that correspond to the DATERANGE tags. The DISCONTINUITY tags mark the following boundaries:

- Where the main content transitions to an ad
- Where one ad transitions to another ad
- Where an ad transitions back to the main content

EXT-X-KEY tags

MediaTailor passes through EXT-X-KEY tags from the input manifest. These tags indicate that the main content is encrypted. Since ads aren't encrypted, MediaTailor inserts EXT-X-KEY:METHOD=NONE at the start of an ad avail. When playback returns to the main content, MediaTailor re-enables encryption by inserting the EXT-X-KEY tag with the METHOD value defined as the encryption type.

Unrecognized tags

MediaTailor passes through all unknown and custom tags from the input manifest to the output manifest.

HLS manifest examples

The following sections provide examples of HLS origin manifests and personalized manifests. Understanding these examples can help you configure and troubleshoot your MediaTailor workflows.

For information about how query parameters are applied to HLS manifests and segments, see [MediaTailor HLS implicit session initialization](#).

Understanding HLS playlist types

HTTP Live Streaming (HLS) uses two primary types of playlists:

Multivariant playlist

A multivariant playlist is the top-level index file that lists all available renditions of the content. It contains references to media playlists but does not contain any media segments itself. This

playlist allows players to select the most appropriate rendition based on network conditions, device capabilities, or user preferences.

This playlist type is also known by several other names in various contexts, including master playlist, master manifest, primary playlist, main playlist, index file, or master M3U8.

In MediaTailor workflows, the multivariant playlist is the entry point for playback requests and is where ad personalization begins.

Media playlist

A media playlist contains the actual media segment information for a specific rendition (quality level) of the content. It includes timing information, segment URLs, and other metadata required for playback of a single rendition.

This playlist type is also known as media playlist, child manifest, chunklist, media M3U8, or rendition playlist.

In MediaTailor workflows, media playlists are personalized to include both content segments and ad segments in the proper sequence.

For more detailed information about HLS playlist types, see [HLS playlist types](#).

HLS origin manifest examples

The following example shows an HLS multivariant playlist that AWS Elemental MediaTailor received by HLS from the content origin.

```
#EXTM3U
#EXT-X-VERSION:3
#EXT-X-INDEPENDENT-SEGMENTS
#EXT-X-STREAM-INF:BANDWIDTH=2665726,AVERAGE-
BANDWIDTH=2526299,RESOLUTION=960x540,FRAME-
RATE=29.970,CODECS="avc1.640029,mp4a.40.2",SUBTITLES="subtitles"
index_1.m3u8
#EXT-X-STREAM-INF:BANDWIDTH=3956044,AVERAGE-
BANDWIDTH=3736264,RESOLUTION=1280x720,FRAME-
RATE=29.970,CODECS="avc1.640029,mp4a.40.2",SUBTITLES="subtitles"
index_2.m3u8
#EXT-X-STREAM-INF:BANDWIDTH=995315,AVERAGE-
BANDWIDTH=951107,RESOLUTION=640x360,FRAME-
RATE=29.970,CODECS="avc1.4D401E,mp4a.40.2",SUBTITLES="subtitles"
index_3.m3u8
```

```
#EXT-X-MEDIA:TYPE=SUBTITLES, GROUP-
ID="subtitles", NAME="caption_1", DEFAULT=YES, AUTOSELECT=YES, FORCED=NO, LANGUAGE="eng", URI="index_
```

In this multivariant playlist example:

- The #EXT-X-STREAM-INF tags define different renditions with varying resolutions and bitrates
- Each rendition references a media playlist (such as index_1.m3u8)
- The #EXT-X-MEDIA tag defines a subtitle track

The following example shows an HLS media playlist that AWS Elemental MediaTailor received by HLS from the content origin. This example uses EXT-X-CUE-OUT and EXT-X-CUE-IN tags to describe ad avail opportunities.

```
#EXTM3U
#EXT-X-VERSION:3
#EXT-X-TARGETDURATION:7
#EXT-X-MEDIA-SEQUENCE:8779957
#EXTINF:6.006,
index_1_8779957.ts?m=1566416212
#EXTINF:6.006,
index_1_8779958.ts?m=1566416212
#EXTINF:5.372,
index_1_8779959.ts?m=1566416212
#EXT-OATCLS-SCTE35:/DA1AAAAAsvhAP/wFAXwAAAGf+AdLfIP4AG3dAAAEBQAAXytxmQ==
#EXT-X-CUE-OUT:20.020
#EXTINF:0.634,
index_1_8779960.ts?m=1566416212
#EXT-X-CUE-OUT-CONT:ElapsedTime=0.634,Duration=21,SCTE35=/DA1AAAAAsvhAP/wFAXwAAAGf
+AdLfIP4AG3dAAAEBQAAXytxmQ==
#EXTINF:6.006,
index_1_8779961.ts?m=1566416212
#EXT-X-CUE-OUT-CONT:ElapsedTime=6.640,Duration=21,SCTE35=/DA1AAAAAsvhAP/wFAXwAAAGf
+AdLfIP4AG3dAAAEBQAAXytxmQ==
#EXTINF:6.006,
index_1_8779962.ts?m=1566416212
#EXT-X-CUE-OUT-CONT:ElapsedTime=12.646,Duration=21,SCTE35=/DA1AAAAAsvhAP/wFAXwAAAGf
+AdLfIP4AG3dAAAEBQAAXytxmQ==
#EXTINF:6.006,
index_1_8779963.ts?m=1566416212
#EXT-X-CUE-OUT-CONT:ElapsedTime=18.652,Duration=21,SCTE35=/DA1AAAAAsvhAP/wFAXwAAAGf
+AdLfIP4AG3dAAAEBQAAXytxmQ==
```



```
#EXTINF:1.368,
index_1_8779964.ts?m=1566416212
#EXT-X-CUE-IN
#EXTINF:4.638,
index_1_8779965.ts?m=1566416212
#EXTINF:6.006,
index_1_8779966.ts?m=1566416212
#EXTINF:6.006,
index_1_8779967.ts?m=1566416212
#EXTINF:6.006,
index_1_8779968.ts?m=1566416212
```

In this media playlist example:

- The #EXTINF tags specify the duration of each segment
- The #EXT-X-CUE-OUT tag marks the beginning of an ad break
- The #EXT-X-CUE-OUT-CONT tags provide information about the ongoing ad break
- The #EXT-X-CUE-IN tag marks the end of the ad break

HLS personalized manifest examples

The following example shows an HLS multivariant playlist that AWS Elemental MediaTailor personalized.

```
#EXTM3U
#EXT-X-VERSION:3
#EXT-X-MEDIA:LANGUAGE="eng",AUTOSELECT=YES,FORCED=NO,TYPE=SUBTITLES,URI="../../../../
manifest/43f3e412052f2808dd84ea1da90e92e914edddde/external-
canary-hls/ee1696a8-4f7f-4c4c-99de-9821131847e8/3.m3u8",GROUP-
ID="subtitles",DEFAULT=YES,NAME="caption_1"
#EXT-X-INDEPENDENT-SEGMENTS
#EXT-X-STREAM-INF:CODECS="avc1.640029,mp4a.40.2",AVERAGE-
BANDWIDTH=2526299,RESOLUTION=960x540,SUBTITLES="subtitles",FRAME-
RATE=29.97,BANDWIDTH=2665726
../../../../manifest/43f3e412052f2808dd84ea1da90e92e914edddde/external-canary-hls/
ee1696a8-4f7f-4c4c-99de-9821131847e8/0.m3u8
#EXT-X-STREAM-INF:CODECS="avc1.640029,mp4a.40.2",AVERAGE-
BANDWIDTH=3736264,RESOLUTION=1280x720,SUBTITLES="subtitles",FRAME-
RATE=29.97,BANDWIDTH=3956044
../../../../manifest/43f3e412052f2808dd84ea1da90e92e914edddde/external-canary-hls/
ee1696a8-4f7f-4c4c-99de-9821131847e8/1.m3u8
```

```
#EXT-X-STREAM-INF:CODECS="avc1.4D401E,mp4a.40.2",AVERAGE-
BANDWIDTH=951107,RESOLUTION=640x360,SUBTITLES="subtitles",FRAME-
RATE=29.97,BANDWIDTH=995315
../.../manifest/43f3e412052f2808dd84ea1da90e92e914eddddee/external-canary-hls/
ee1696a8-4f7f-4c4c-99de-9821131847e8/2.m3u8
```

Notice how MediaTailor has modified the media playlist URLs to include session-specific information that enables personalized ad insertion.

The following example shows a media playlist that AWS Elemental MediaTailor personalized.

```
#EXTM3U
#EXT-X-VERSION:6
#EXT-X-TARGETDURATION:7
#EXT-X-MEDIA-SEQUENCE:8779957
#EXT-X-DISCONTINUITY-SEQUENCE:0
#EXTINF:6.006,
https://777788889999.mediapackage.us-west-2.amazonaws.com/out/v1/
e309ffd02ba8498d864dcaacff7a5ad9/index_1_8779957.ts?m=1566416212
#EXTINF:6.006,
https://777788889999.mediapackage.us-west-2.amazonaws.com/out/v1/
e309ffd02ba8498d864dcaacff7a5ad9/index_1_8779958.ts?m=1566416212
#EXTINF:5.372,
https://777788889999.mediapackage.us-west-2.amazonaws.com/out/v1/
e309ffd02ba8498d864dcaacff7a5ad9/index_1_8779959.ts?m=1566416212
#EXT-X-DISCONTINUITY
#EXTINF:3.066667,
../.../.../segment/43f3e412052f2808dd84ea1da90e92e914eddddee/external-canary-hls/
ee1696a8-4f7f-4c4c-99de-9821131847e8/0/8779960
#EXTINF:3.0,
../.../.../segment/43f3e412052f2808dd84ea1da90e92e914eddddee/external-canary-hls/
ee1696a8-4f7f-4c4c-99de-9821131847e8/0/8779961
#EXTINF:3.0,
../.../.../segment/43f3e412052f2808dd84ea1da90e92e914eddddee/external-canary-hls/
ee1696a8-4f7f-4c4c-99de-9821131847e8/0/8779962
#EXTINF:3.0,
../.../.../segment/43f3e412052f2808dd84ea1da90e92e914eddddee/external-canary-hls/
ee1696a8-4f7f-4c4c-99de-9821131847e8/0/8779963
#EXTINF:2.966667,
../.../.../segment/43f3e412052f2808dd84ea1da90e92e914eddddee/external-canary-hls/
ee1696a8-4f7f-4c4c-99de-9821131847e8/0/8779964
#EXT-X-DISCONTINUITY
#EXTINF:6.006,
```

```

https://777788889999.mediapackage.us-west-2.amazonaws.com/out/v1/
e309ffd02ba8498d864dcaacff7a5ad9/index_1_8779963.ts?m=1566416212
#EXTINF:1.368,
https://777788889999.mediapackage.us-west-2.amazonaws.com/out/v1/
e309ffd02ba8498d864dcaacff7a5ad9/index_1_8779964.ts?m=1566416212
#EXTINF:4.638,
https://777788889999.mediapackage.us-west-2.amazonaws.com/out/v1/
e309ffd02ba8498d864dcaacff7a5ad9/index_1_8779965.ts?m=1566416212
#EXTINF:6.006,
https://777788889999.mediapackage.us-west-2.amazonaws.com/out/v1/
e309ffd02ba8498d864dcaacff7a5ad9/index_1_8779966.ts?m=1566416212
#EXTINF:6.006,
https://777788889999.mediapackage.us-west-2.amazonaws.com/out/v1/
e309ffd02ba8498d864dcaacff7a5ad9/index_1_8779967.ts?m=1566416212
#EXTINF:6.006,
https://777788889999.mediapackage.us-west-2.amazonaws.com/out/v1/
e309ffd02ba8498d864dcaacff7a5ad9/index_1_8779968.ts?m=1566416212

```

In this personalized media playlist example:

- MediaTailor has inserted ad segments between the content segments
- The #EXT-X-DISCONTINUITY tags mark the transitions between content and ads
- Content segments are served from the origin server (MediaPackage in this example)
- Ad segments are served from MediaTailor's ad segment storage

Key differences in personalized manifests

When MediaTailor personalizes HLS manifests, it makes several important changes:

Multivariant playlist changes

- Variant playlist URLs are rewritten to point to MediaTailor-managed URLs that include session information
- The order of tags may be reorganized for optimal playback

Variant playlist changes

- Ad markers (EXT-X-CUE-OUT, EXT-X-CUE-IN) are replaced with actual ad segments
- Discontinuity markers (EXT-X-DISCONTINUITY) are added at content/ad boundaries
- Content segment URLs are rewritten to point to the origin or CDN
- Ad segment URLs are added to point to MediaTailor's ad segment storage

Understanding these changes can help you troubleshoot issues in your MediaTailor workflows and ensure proper configuration of your CDN and player.

Related topics

For more information about HLS manifests and MediaTailor, see the following topics:

- [HLS playlist types](#) - Detailed explanation of HLS playlist types
- [Using a CDN to optimize MediaTailor ad personalization and content delivery](#) - Information about using a CDN with MediaTailor
- [How MediaTailor ad insertion works](#) - Overview of how MediaTailor ad insertion works

Integrating an MPEG-DASH source

AWS Elemental MediaTailor supports .mpd live and video on demand (VOD) manifests that follow the guidelines for the DASH dynamic profile. MediaTailor accepts multi-period and single-period DASH-compliant manifest inputs, and delivers multi-period DASH-compliant manifest outputs.

Input manifests must have the following:

- SCTE-35 event streams with splice info settings for either `splice insert` or `time signal`. The settings can be provided in clear XML or in base64-encoded binary.
- `Segment templates` with `segment timelines`.

For published manifests, MediaTailor requires that updates from the origin server leave the following unchanged:

- Period start times, specified in the `start` attribute.
- Values of `presentationTimeOffset` in the segment templates of the period representations.

As a best practice, give the ad avails the same `AdaptationSet` and `Representation` settings as the content stream periods. AWS Elemental MediaTailor uses these settings to transcode the ads to match the content stream, for smooth switching between the two.

The following sections provide more information about how MediaTailor handles ads in DASH manifests.

Topics

- [DASH ad markers](#)
- [DASH ad avail duration](#)
- [DASH manifest segment numbering](#)
- [DASH MPD examples](#)
- [DASH location feature](#)

DASH ad markers

MediaTailor

AWS Elemental MediaTailor uses SCTE-35 cue-out markers to identify ad avails in the DASH manifest using the following logic:

- **Multi-period DASH:** MediaTailor inserts ads for the first Event in each Period that contains either `SpliceInsert` or `TimeSignal` cue-out markers. MediaTailor ignores additional Event markers in the Period.
- **Single-period DASH:** MediaTailor inserts ads each Event in the Period that contains either `SpliceInsert` or `TimeSignal` cue-out markers.

By default, AWS Elemental MediaTailor manages DASH manifests as multi-period manifests. You can change your configuration to handle single-period DASH manifests from your origin server. For information, see [the section called “Creating a configuration”](#).

The following sections provide additional detail about DASH ad marker handling and provides decorated manifests from the origin.

DASH origin manifest XML requirements

Ad markers in DASH manifests from the origin must be formatted properly for MediaTailor to identify ad breaks. The following topics describe these formatting requirements in clear XML.

SpliceInsert in clear XML

`SpliceInsert` ad markers in clear XML must contain the following:

- `EventStream` must have the following attribute:
`schemeIdUri=urn:scte:scte35:2013:xml`
- Event must hold `scte35:SpliceInfoSection`

- `scte35:SpliceInfoSection` must hold `scte35:SpliceInsert`
- `scte35:SpliceInsert` must have the following attribute:
`outOfNetworkIndicator="true"`

Example SpliceInsert in XML

In the following example, required SCTE markers are in bold.

```
<Period start="PT444806.040S" id="123586" duration="PT15.000S">
  <EventStream timescale="90000" schemeIdUri="urn:scte:scte35:2013:xml">
    <Event duration="1350000">
      <scte35:SpliceInfoSection protocolVersion="0" ptsAdjustment="180832" tier="4095">
        <scte35:SpliceInsert spliceEventId="4026531855"
spliceEventCancelIndicator="false" outOfNetworkIndicator="true"
spliceImmediateFlag="false" uniqueProgramId="1" availNum="1" availsExpected="1">
          <scte35:Program><scte35:SpliceTime ptsTime="5672624400"/></scte35:Program>
          <scte35:BreakDuration autoReturn="true" duration="1350000"/>
        </scte35:SpliceInsert>
      </scte35:SpliceInfoSection>
    </Event>
    .
    .
    .
  </EventStream>
</Period>
```

TimeSignal in clear XML

TimeSignal ad markers in clear XML must contain the following:

- EventStream must have the following attribute:
`schemeIdUri=urn:scte:scte35:2013:xml`
- Event must hold `scte35:SpliceInfoSection`
- `scte35:SpliceInfoSection` must hold `scte35:TimeSignal`
- `scte35:SpliceInfoSection` must also hold `scte35:SegmentationDescriptor`
- `scte35:SegmentationDescriptor` must have the following attribute, where the value is a valid [Cue-out numbers](#): `segmentationTypeId="xx"`
- `scte35:SegmentationDescriptor` must hold `scte35:SegmentationUpid`

Cue-out numbers

The following are supported cue-out numbers for the `segmentationTypeId`.

Segmentation message	segmentationTypeId value	Hexadecimal value
Distributor advertisement end	51	0x51
Distributor advertisement start	50	0x32
Distributor placement opportunity end	55	0x37
Distributor placement opportunity start	54	0x36
End break	35	0x23
Provider advertisement end	49	0x31
Provider advertisement start	48	0x30
Provider overlay placement opportunity end	57	0x39
Provider overlay placement opportunity start	56	0x38
Provider placement opportunity end	53	0x35
Provider placement opportunity start	52	0x34
Start break	34	0x22

Example TimeSignal in XML

In the following example, required SCTE markers are in bold.

```

<Period start="PT346530.250S" id="178443" duration="PT61.561S">
  <EventStream timescale="90000" schemeIdUri="urn:scte:scte35:2013:xml">
    <Event duration="5310000">
      <scte35:SpliceInfoSection protocolVersion="0" ptsAdjustment="183003" tier="4095">
        <scte35:TimeSignal>
          <scte35:SpliceTime ptsTime="3442857000"/>
        </scte35:TimeSignal>
        <scte35:SegmentationDescriptor segmentationEventId="1414668"
segmentationEventCancelIndicator="false"
segmentationDuration="8100000" segmentationTypeId="52" segmentNum="0"
segmentsExpected="0">
          <scte35:DeliveryRestrictions webDeliveryAllowedFlag="false"
noRegionalBlackoutFlag="false" archiveAllowedFlag="false" deviceRestrictions="3"/>
          <scte35:SegmentationUpid segmentationUpidType="12"
segmentationUpidLength="2">0100</scte35:SegmentationUpid>
        </scte35:SegmentationDescriptor>
      </scte35:SpliceInfoSection>
    </Event>
    .
    .
    .
  </EventStream>
</Period>

```

DASH origin manifest base64-encoded binary requirements

Ad markers in DASH manifests from the origin must be formatted properly for MediaTailor to identify ad breaks. The following topics describe these formatting requirements in base64-encoded binary.

Both TimeSignal and SpliceInsert ad markers in base64-encoded manifests must contain the following:

- EventStream must have the following attribute: `urn:scte:scte35:2014:xml+bin`
- Event must hold `scte35:Signal`
- `scte35:Signal` must hold `scte35:Binary` that contains a base64-encoded binary.

The decoded binary must provide a `splice_info_section` with the same information as what's required for clear XML ad markers.

- The command type must be either `splice_insert()` or `time_signal()`

- The additional settings must comply with those described in [TimeSignal in clear XML](#) and [SpliceInsert in clear XML](#).

The decoded binary must provide a `splice_info_section` with the same set of information as the clear XML would provide in a `scte35:SpliceInfoSection` element. The command type must be either `splice_insert()` or `time_signal()`, and the additional settings must comply with those described previously for clear XML delivery.

The following example shows this option, with the required markers in bold.

```
<Period start="PT444806.040S" id="123586" duration="PT15.000S">
  <EventStream schemeIdUri="urn:scte:scte35:2014:xml+bin" timescale="1">
    <Event presentationTime="1541436240" duration="24" id="29">
      <scte35:Signal xmlns="http://www.scte.org/schemas/35/2016">
        <scte35:Binary>/DAhAAAAAAAAAAP/wEAUAAAHaf+9/fgAg9YDAAAAAAAAA25aoh</
scte35:Binary>
      </scte35:Signal>
    </Event>
    <Event presentationTime="1541436360" duration="24" id="30">
      <scte35:Signal xmlns="http://www.scte.org/schemas/35/2016">

        <scte35:Binary>QW5vdGhlciB0ZXN0IHN0cmVuZyBmb3IgZW5jb2RpbmcgdG8gQmFzZTY0IGVuY29kZWQgYm1uYXJ5Lg=
scte35:Binary>
        </scte35:Signal>
      </Event>
      .
      .
      .
    </Period>
```

The following is the decoded binary for the first event listed in the preceding example. The setting for `splice_command_type` is 5, which indicates `splice_insert`.

```
{
  "table_id": 252,
  "section_syntax_indicator": false,
  "private_indicator": false,
  "section_length": 33,
  "protocol_version": 0,
  "encrypted_packet": false,
  "encryption_algorithm": 0,
```

```

    "pts_adjustment": 0,
    "cw_index": 0,
    "tier": "0xFFFF",
    "splice_command_length": 16,
    "splice_command_type": 5,
    "splice_command": {
        "splice_event_id": 448,
        "splice_event_cancel_indicator": false,
        "out_of_network_indicator": true,
        "program_splice_flag": true,
        "duration_flag": true,
        "splice_immediate_flag": false,
        "utc_splice_time": {
            "time_specified_flag": false,
            "pts_time": null
        },
    },
    "component_count": 0,
    "components": null,
    "break_duration": {
        "auto_return": false,
        "duration": {
            "pts_time": 2160000,
            "wall_clock_seconds": 24.0,
            "wall_clock_time": "00:00:24:00000"
        }
    },
    },
    "unique_program_id": 49152,
    "avail_num": 0,
    "avails_expected": 0
},
"splice_descriptor_loop_length": 0,
"splice_descriptors": null,
"Scte35Exception": {
    "parse_status": "SCTE-35 cue parsing completed with 0 errors.",
    "error_messages": [],
    "table_id": 252,
    "splice_command_type": 5
}
}

```

DASH ad avail duration

During playback, when AWS Elemental MediaTailor encounters an ad avail, it replaces some or all of the avail with ads. MediaTailor starts ad replacement at the beginning of the ad avail and includes ads as follows:

- If the ad avail specifies a duration, MediaTailor includes as many ads as it can fit inside the duration boundary, without overwriting content that follows.
- If no duration is provided, MediaTailor includes ads until it reaches the end of the ad avail. For multi-period manifests, this is the end of the period. For single-period manifests, this is the end of the event. MediaTailor doesn't play ads past the end of the ad avail and, when it encounters the end, truncates the current ad instead of overwriting the content that follows.

How AWS Elemental MediaTailor looks for the ad avail duration

AWS Elemental MediaTailor searches for a duration setting in the following order:

1. Event duration
2. For splice insert, the `scte35:BreakDuration` duration
3. For time signal, the `scte35:SegmentationDescriptor` `segmentationDuration`

If AWS Elemental MediaTailor doesn't find any of these settings, it manages ad inclusion without a duration.

The following example shows an Event that has a duration.

```
<Period start="PT444806.040S" id="123586" duration="PT15.000S">
  <EventStream timescale="90000" schemeIdUri="urn:scte:scte35:2013:xml">
    <Event duration="1350000">
      <scte35:SpliceInfoSection protocolVersion="0" ptsAdjustment="180832"
tier="4095">
        <scte35:SpliceInsert spliceEventId="4026531855"
spliceEventCancelIndicator="false" outOfNetworkIndicator="true"
spliceImmediateFlag="false" uniqueProgramId="1" availNum="1" availsExpected="1">
          <scte35:Program><scte35:SpliceTime ptsTime="5672624400"/></
scte35:Program>
            <scte35:BreakDuration autoReturn="true" duration="1350000"/>
          </scte35:SpliceInsert>
        </scte35:SpliceInfoSection>
```

```

</Event>
...

```

The following example shows ad avail with no duration specified. The Event has no duration and the `scte35:SpliceInsert` element doesn't contain a `scte35:BreakDuration` child element.

```

<Period start="PT444836.720S" id="123597" duration="PT12.280S">
  <EventStream timescale="90000" schemeIdUri="urn:scte:scte35:2013:xml">
    <Event>
      <scte35:SpliceInfoSection protocolVersion="0" ptsAdjustment="180832"
tier="4095">
        <scte35:SpliceInsert spliceEventId="4026531856"
spliceEventCancelIndicator="false" outOfNetworkIndicator="true"
spliceImmediateFlag="false" uniqueProgramId="1" availNum="1" availsExpected="1">
          <scte35:Program><scte35:SpliceTime ptsTime="5675385600"/></
scte35:Program>
        </scte35:SpliceInsert>
      </scte35:SpliceInfoSection>
    </Event>
    ...
  </EventStream>
</Period>

```

DASH manifest segment numbering

MediaTailor supports media segments in `<SegmentTemplate>` that are defined using `<SegmentTimeline>` and the media attribute. You can specify the media segment list in the media attribute using either the `$Number$` identifier or the `$Time$` identifier.

The following example shows a `SegmentTemplate` with a media attribute setting that uses the `$Number$` identifier.

```

<SegmentTemplate initialization="index_subtitles_4_0_init.mp4?
m=1532451703" media="index_subtitles_4_0_$Number$.mp4?m=1532451703"
presentationTimeOffset="1062336677920" startNumber="2349899" timescale="90000">
  <SegmentTimeline>
    <S d="540540" r="2" t="1062338840080"/>
    <S d="69069" t="1062340461700"/>
  </SegmentTimeline>
</SegmentTemplate>

```

The following example shows a `SegmentTemplate` with a media attribute setting that uses the `$Time$` identifier.

```

<SegmentTemplate
initialization="asset_720p_8000K_9_init.mp4" media="asset_720p_8000K_9_$Time$.mp4"
startNumber="1" timescale="90000">
  <SegmentTimeline>
    <S d="180000" r="2" t="0"/>
    <S d="147000" t="540000"/>
  </SegmentTimeline>
</SegmentTemplate>

```

DASH MPD examples

The following sections provide examples of DASH origin MPDs and personalized MPDs. Understanding these examples can help you configure and troubleshoot your MediaTailor workflows.

For information about how query parameters are applied to DASH manifests and segments, see [MediaTailor DASH implicit session initialization](#).

Understanding DASH MPD structure

Dynamic Adaptive Streaming over HTTP (DASH) uses a Media Presentation Description (MPD) manifest to deliver streaming content. The MPD is an XML document that describes the structure and availability of media content.

MPD (Media Presentation Description)

The MPD is the primary manifest file in DASH streaming that describes the structure and availability of media content. It contains information about periods, adaptation sets, representations, and segments that make up the streaming content.

This manifest type is also known by several other names in various contexts, including DASH manifest, DASH MPD, master manifest (when comparing to HLS), or presentation manifest.

In MediaTailor workflows, the MPD is the entry point for playback requests and is where ad personalization begins.

Period

A Period is a temporal section of a DASH presentation. Each Period contains one or more adaptation sets and represents a span of media time. In ad insertion workflows, separate Periods are typically used to delineate between content and ads.

In MediaTailor workflows, Periods are used to separate main content from ad content, with each ad typically represented by its own Period.

AdaptationSet

An AdaptationSet groups a set of interchangeable encoded versions of one or several media content components. For example, one AdaptationSet might contain multiple video quality levels, while another might contain multiple audio language options.

In MediaTailor workflows, AdaptationSets are preserved during ad insertion to maintain consistent media types between content and ads.

Representation

A Representation is a specific encoded version of the media content within an AdaptationSet. Each Representation typically differs in bitrate, resolution, or other encoding parameters, allowing clients to select the most appropriate version based on network conditions and device capabilities.

In MediaTailor workflows, Representations in ad Periods are matched as closely as possible to the Representations in content Periods to ensure a smooth viewing experience.

For more detailed information about DASH manifest types, see [DASH manifest types](#).

For information about DASH manifest structure and configuration in AWS Elemental MediaPackage, see the MediaPackage User Guide section on DASH overview.

Live DASH MPD examples

This section provides examples of live DASH MPDs. Each example lists an MPD as received from the origin server and after MediaTailor has personalized the MPD with ads.

DASH MPD splice insert example

DASH origin MPD example for splice insert

The following example from an MPD shows an ad avail in a manifest received by DASH from the content origin. This example uses the SpliceInsert marker to indicate an ad avail.

```
<Period start="PT173402.036S" id="46041">
  <EventStream timescale="90000" schemeIdUri="urn:scte:scte35:2013:xml">
    <Event duration="9450000">
```

```

    <scte35:SpliceInfoSection protocolVersion="0" ptsAdjustment="183265" tier="4095">
      <scte35:SpliceInsert spliceEventId="99" spliceEventCancelIndicator="false"
outOfNetworkIndicator="true" spliceImmediateFlag="false" uniqueProgramId="1"
availNum="1" availsExpected="1">
        <scte35:Program><scte35:SpliceTime ptsTime="7835775000"/></scte35:Program>
        <scte35:BreakDuration autoReturn="true" duration="9450000"/>
      </scte35:SpliceInsert>
      <scte35:SegmentationDescriptor segmentationEventId="99"
segmentationEventCancelIndicator="false" segmentationDuration="9450000">
        <scte35:DeliveryRestrictions webDeliveryAllowedFlag="true"
noRegionalBlackoutFlag="true" archiveAllowedFlag="true" deviceRestrictions="3"/>
        <scte35:SegmentationUpid segmentationUpidType="8" segmentationUpidLength="0"/
>
        <scte35:SegmentationTypeID segmentationType="52"/>
        <scte35:SegmentNum segmentNum="1"/>
        <scte35:SegmentsExpected segmentsExpected="1"/>
      </scte35:SegmentationDescriptor>
    </scte35:SpliceInfoSection>
  </Event>
</EventStream>
<AdaptationSet mimeType="video/mp4" segmentAlignment="true"
subsegmentAlignment="true" startWithSAP="1" subsegmentStartsWithSAP="1"
bitstreamSwitching="true">
  <Representation id="1" width="960" height="540" frameRate="30000/1001"
bandwidth="1000000" codecs="avc1.4D401F">
    <SegmentTemplate timescale="30000" media="index_video_1_0_$Number
$.mp4?m=1528475245" initialization="index_video_1_0_init.mp4?m=1528475245"
startNumber="178444" presentationTimeOffset="10395907501">
      <SegmentTimeline>
        <S t="10395907501" d="60060" r="29"/>
        <S t="10397709301" d="45045"/>
      </SegmentTimeline>
    </SegmentTemplate>
  </Representation>
</AdaptationSet>
<AdaptationSet mimeType="audio/mp4" segmentAlignment="0" lang="eng">
  <Representation id="2" bandwidth="96964" audioSamplingRate="48000"
codecs="mp4a.40.2">
    <SegmentTemplate timescale="48000" media="index_audio_2_0_$Number
$.mp4?m=1528475245" initialization="index_audio_2_0_init.mp4?m=1528475245"
startNumber="178444" presentationTimeOffset="16633452001">
      <SegmentTimeline>
        <S t="16633452289" d="96256" r="3"/>
        <S t="16633837313" d="95232"/>
      </SegmentTimeline>
    </SegmentTemplate>
  </Representation>
</AdaptationSet>

```

```

        <S t="16633932545" d="96256" r="4"/>
        <S t="16634413825" d="95232"/>
        <S t="16634509057" d="96256" r="5"/>
        <S t="16635086593" d="95232"/>
        <S t="16635181825" d="96256" r="4"/>
        <S t="16635663105" d="95232"/>
        <S t="16635758337" d="96256" r="5"/>
        <S t="16636335873" d="71680"/>
    </SegmentTimeline>
</SegmentTemplate>
</Representation>
</AdaptationSet>
</Period>

```

In this origin MPD example:

- The <EventStream> element contains SCTE-35 markers that indicate ad avails
- The <scte35:SpliceInsert> element provides details about the ad avail
- The <scte35:BreakDuration> element specifies the duration of the ad break
- The <AdaptationSet> elements define the available video and audio streams

DASH personalized MPD example for splice insert

AWS Elemental MediaTailor personalizes the ad avails with advertising specifications. The personalizations reflect the viewer data that is received from the player and the advertising campaigns that are currently underway.

The following example shows an ad avail after AWS Elemental MediaTailor personalizes it.

```

<Period id="178443_1" start="PT96H15M30.25S">
  <BaseURL>http://111122223333.cloudfront.net/nbc_fallback_2/</BaseURL>
  <AdaptationSet bitstreamSwitching="false" frameRate="30/1" mimeType="video/
mp4" segmentAlignment="true" startWithSAP="1" subsegmentAlignment="true"
subsegmentStartsWithSAP="1">
    <SegmentTemplate startNumber="1" timescale="90000"/>
    <Representation bandwidth="10000000" codecs="avc1.640028" height="1080" id="1"
width="1920">
      <SegmentTemplate initialization="nbc_fallback_ad_2_1080p_10init.mp4"
media="nbc_fallback_ad_2_1080p_10_${Number%09d$.mp4" startNumber="1" timescale="90000">
        <SegmentTimeline>
          <S d="180000" r="13" t="0"/>

```



```

        <S d="176940" t="2520000"/>
    </SegmentTimeline>
</SegmentTemplate>
</Representation>
<Representation bandwidth="4000000" codecs="avc1.64001f" height="720" id="2"
width="1280">
    <SegmentTemplate initialization="nbc_fallback_ad_2_720p_9init.mp4"
media="nbc_fallback_ad_2_720p_9_{$Number%09d$.mp4" startNumber="1" timescale="90000">
        <SegmentTimeline>
            <S d="180000" r="13" t="0"/>
            <S d="176940" t="2520000"/>
        </SegmentTimeline>
    </SegmentTemplate>
</Representation>
</AdaptationSet>
<AdaptationSet mimeType="audio/mp4" segmentAlignment="0" lang="eng">
    <Representation id="8" bandwidth="128000" audioSamplingRate="48000"
codecs="mp4a.40.2">
        <SegmentTemplate initialization="nbc_fallback_ad_2_audio_2init.mp4"
media="nbc_fallback_ad_2_audio_2_{$Number%09d$.mp4" startNumber="1" timescale="90000">
            <SegmentTimeline>
                <S d="180000" r="13" t="0"/>
                <S d="176940" t="2520000"/>
            </SegmentTimeline>
        </SegmentTemplate>
    </Representation>
</AdaptationSet>
</Period>

```

In this personalized MPD example:

- MediaTailor has created a new Period for the ad content
- The <BaseURL> element points to the ad content location
- The <AdaptationSet> elements maintain similar structure to the content
- The <Representation> elements provide different quality levels for the ad content

DASH MPD time signal example

DASH origin MPD example for time signal

The following example from an MPD shows an ad avail in a manifest received by DASH from the content origin. This example uses the `TimeSignal` marker to indicate an ad avail.

```
<Period start="PT173402.036S" id="46041">
  <EventStream timescale="90000" schemeIdUri="urn:scte:scte35:2013:xml">
    <Event duration="9450000">
      <scte35:SpliceInfoSection protocolVersion="0" ptsAdjustment="183265" tier="4095">
        <scte35:TimeSignal>
          <scte35:SpliceTime ptsTime="7835775000"/>
        </scte35:TimeSignal>
        <scte35:SegmentationDescriptor segmentationEventId="99"
segmentationEventCancelIndicator="false" segmentationDuration="9450000">
          <scte35:DeliveryRestrictions webDeliveryAllowedFlag="true"
noRegionalBlackoutFlag="true" archiveAllowedFlag="true" deviceRestrictions="3"/>
          <scte35:SegmentationUpid segmentationUpidType="8" segmentationUpidLength="0"/>
        </scte35:SegmentationDescriptor>
      </scte35:SpliceInfoSection>
    </Event>
  </EventStream>
  <AdaptationSet mimeType="video/mp4" segmentAlignment="true"
subsegmentAlignment="true" startWithSAP="1" subsegmentStartsWithSAP="1"
bitstreamSwitching="true">
    <Representation id="1" width="960" height="540" frameRate="30000/1001"
bandwidth="1000000" codecs="avc1.4D401F">
      <SegmentTemplate timescale="30000" media="index_video_1_0_$Number
$.mp4?m=1528475245" initialization="index_video_1_0_init.mp4?m=1528475245"
startNumber="178444" presentationTimeOffset="10395907501">
        <SegmentTimeline>
          <S t="10395907501" d="60060" r="29"/>
          <S t="10397709301" d="45045"/>
        </SegmentTimeline>
      </SegmentTemplate>
    </Representation>
  </AdaptationSet>
</Period>
```

In this origin MPD example:

- The `<scte35:TimeSignal>` element is used instead of `<scte35:SpliceInsert>`
- The `<scte35:SegmentationDescriptor>` provides additional information about the ad avail

DASH personalized MPD example for time signal

AWS Elemental MediaTailor personalizes the ad avails with advertising specifications. The personalizations reflect the viewer data that is received from the player and the advertising campaigns that are currently underway.

The following example shows an ad avail after AWS Elemental MediaTailor personalizes it.

```
<Period id="178443_1" start="PT96H15M30.25S">
  <BaseURL>http://111122223333.cloudfront.net/nbc_fallback_2/</BaseURL>
  <AdaptationSet bitstreamSwitching="false" frameRate="30/1" mimeType="video/
mp4" segmentAlignment="true" startWithSAP="1" subsegmentAlignment="true"
subsegmentStartsWithSAP="1">
    <SegmentTemplate startNumber="1" timescale="90000"/>
    <Representation bandwidth="10000000" codecs="avc1.640028" height="1080" id="1"
width="1920">
      <SegmentTemplate initialization="nbc_fallback_ad_2_1080p_10init.mp4"
media="nbc_fallback_ad_2_1080p_10_${Number%09d$.mp4" startNumber="1" timescale="90000">
        <SegmentTimeline>
          <S d="180000" r="13" t="0"/>
          <S d="176940" t="2520000"/>
        </SegmentTimeline>
      </SegmentTemplate>
    </Representation>
  </AdaptationSet>
</Period>
```

The personalized MPD for time signal is similar to the one for splice insert, with MediaTailor creating a new Period for the ad content.

VOD DASH MPD examples

This section provides examples of video on demand (VOD) DASH MPDs. Each example lists an MPD as received from the origin server and after MediaTailor has personalized the MPD with ads.

VOD DASH MPDs follow the same structure as live MPDs, but they typically have a `type="static"` attribute in the MPD element and may contain multiple Periods for different content segments.

For examples of VOD DASH MPDs, see the MediaTailor documentation on [DASH ad markers](#).

Key differences in personalized MPDs

When MediaTailor personalizes DASH MPDs, it makes several important changes:

Period handling

- New Periods are created for ad content
- Period start times are adjusted to maintain timeline continuity
- EventStream elements with SCTE-35 markers are processed and removed

AdaptationSet and Representation handling

- AdaptationSets in ad Periods are created to match content AdaptationSets
- Representations are created for different quality levels of the ad content
- SegmentTemplate elements are updated to point to ad content

Understanding these changes can help you troubleshoot issues in your MediaTailor workflows and ensure proper configuration of your CDN and player.

Related topics

For more information about DASH MPDs and MediaTailor, see the following topics:

- [DASH manifest types](#) - Detailed explanation of DASH manifest types
- [Using a CDN to optimize MediaTailor ad personalization and content delivery](#) - Information about using a CDN with MediaTailor
- [How MediaTailor ad insertion works](#) - Overview of how MediaTailor ad insertion works
- For comprehensive information about DASH manifest structure and MediaPackage configuration, see the MediaPackage User Guide section on DASH overview.

DASH location feature

This section provides information about the location feature for DASH, which is enabled by default in AWS Elemental MediaTailor. Read this section if you create content delivery network (CDN) routing rules for accessing MediaTailor manifests. Also read this section if you use server-side reporting with players that don't support sticky HTTP redirects.

What is the location feature?

The location feature allows players that don't support sticky HTTP redirects to provide sticky behavior in their manifest update requests.

AWS Elemental MediaTailor uses sessionless initialization, and it requires sticky HTTP redirect behavior from its players. With server-side reporting, when the player makes a request for a manifest update to MediaTailor, the service issues a 302 temporary redirect, to direct the player to an endpoint for the personalized manifest. MediaTailor includes a session ID in the response, as a query parameter. The intent is for the player to follow the URL for the entirety of the session, but players that don't support sticky HTTP redirects drop the redirect and return to the original URL. When a player returns to the original URL, for each new request MediaTailor creates a new session rather than staying with the original session. This can cause the manifest to become corrupt.

The DASH specification provides a solution to this problem in the location feature, which is enabled by default in AWS Elemental MediaTailor configurations. When this feature is enabled, MediaTailor puts the absolute URL in the manifest <Location> tag. Players that don't support sticky HTTP redirects can use the URL provided in <Location> to request updates to the manifest.

Do I need to disable the location feature in my configuration?

The location feature overrides any CDN routing rules that you set up for accessing AWS Elemental MediaTailor manifests, so you might need to disable it. The location feature doesn't affect CDN caching of content or ad segments.

Find your situation in the following list to determine whether you need to disable the location feature for your configuration and how to handle it:

- If you don't have CDN routing rules set up for accessing AWS Elemental MediaTailor manifests, leave the location setting enabled.
- Otherwise, use the following rules:
 - If you either don't use server-side reporting or your players all support sticky HTTP redirects, disable the location feature. For information about how to do this on the console, see [the section called "Creating a configuration"](#).
 - Otherwise, contact [AWS Support](#).

Do I need to use the location feature?

You need to use the location feature for players that don't support sticky HTTP redirects. Use the URL provided in the <Location> tag for all of your manifest update requests.

Example

Example URLs and example <Location> tag.

- **Example: Initial request URL**

```
https://777788889999.mediatailor.us-east-1.amazonaws.com/v1/  
dash/5ca4c1892b1f213a1247fad47b3e34c454a7d490/testLocationTag/index.mpd
```

- **Example: Redirected 302 response**

```
/v1/dash/5ca4c1892b1f213a1247fad47b3e34c454a7d490/testLocationTag/index.mpd?  
aws.sessionId=0e5d9b45-ae97-49eb-901b-893d043e0aa6
```

- **Example: Location tag in a manifest**

```
<Location>https://777788889999.mediatailor.us-east-1.amazonaws.com/v1/  
dash/5ca4c1892b1f213a1247fad47b3e34c454a7d490/testLocationTag/index.mpd?  
aws.sessionId=0e5d9b45-ae97-49eb-901b-893d043e0aa6</Location>
```

Securing AWS Elemental MediaTailor origin interactions with SigV4

Signature Version 4 (SigV4) is a signing protocol used to authenticate MediaTailor requests to supported origins over HTTPS. MediaTailor only supports HTTPS communication and does not allow HTTP connections. With SigV4 signing, MediaTailor includes a signed authorization header in the HTTPS origin request to MediaTailor Channel Assembly, Amazon S3 and AWS Elemental MediaPackage version 2.

You can use SigV4 at your origin to ensure that manifest requests are only fulfilled if they're from MediaTailor and contain a signed authorization header. This way, unauthorized MediaTailor playback configurations are blocked from accessing your origin content. If the signed authorization header is valid, your origin fulfills the request. If it isn't valid, the request fails.

The following sections describe requirements for using MediaTailor SigV4 signing to supported origins.

MediaTailor Channel Assembly requirements

If you use SigV4 to protect your MediaTailor Channel Assembly origin, the following requirements must be met for MediaTailor to access the manifest:

- The origin base URL in your MediaTailor configuration must be a Channel Assembly channel in the following format: `channel-assembly.mediatailor.region.amazonaws.com`
- Your origin must be configured to use HTTPS. MediaTailor only supports HTTPS communication and does not allow HTTP connections. If HTTPS is not enabled at the origin, MediaTailor will not sign the request.
- Your channel must have an origin access policy that includes the following:
 - Principal access for MediaTailor to access your channel. Grant access to **mediataylor.amazonaws.com**.
 - IAM permissions **mediataylor:GetManifest** to read all multivariant playlists referenced by the MediaTailor configuration.

For information about setting a policy on the channel, see [Create a channel using the MediaTailor console](#).

Example origin access policy for Channel Assembly, scoped to the MediaTailor configuration account

```
{
  "Effect": "Allow",
  "Principal": {"Service": "mediataylor.amazonaws.com"},
  "Action": "mediataylor:GetManifest",
  "Resource": "arn:aws:mediataylor:us-west-2:777788889999:channel/ca-origin-channel",
  "Condition": {
    "StringEquals": {"AWS:SourceAccount": "777788889999"}
  }
}
```

Example origin access policy for Channel Assembly, scoped to the MediaTailor playback configuration

```
{
  "Effect": "Allow",
  "Principal": {"Service": "mediataylor.amazonaws.com"},
  "Action": "mediataylor:GetManifest",
  "Resource": "arn:aws:mediataylor:us-west-2:777788889999:channel/ca-origin-channel",
  "Condition": {
    "StringEquals": {"AWS:SourceArn": "arn:aws:mediataylor:us-west-2:777788889999:playbackConfiguration/test"}
  }
}
```

```
}
```

Amazon S3 requirements

If you use SigV4 to protect your Amazon S3 origin, the following requirements must be met for MediaTailor to access the manifest:

- The origin base URL in your MediaTailor configuration must be an S3 bucket in the following format: `s3.region.amazonaws.com`
- Your origin must be configured to use HTTPS. MediaTailor only supports HTTPS communication and does not allow HTTP connections. If HTTPS is not enabled at the origin, MediaTailor will not sign the request.
- Your channel must have an origin access policy that includes the following:
 - Principal access for MediaTailor to access your bucket. Grant access to **mediatailor.amazonaws.com**.

For information about configuring access in IAM, see [Access management](#) in the *AWS Identity and Access Management User Guide*.

- IAM permissions **s3:GetObject** to read all top-level manifests referenced by the MediaTailor configuration.

For general information about SigV4 for Amazon S3, see the [Authenticating Requests \(AWS Signature Version 4\)](#) topic in the *Amazon S3 API reference*.

Example origin access policy for Amazon S3, scoped to the MediaTailor account

```
{
  "Effect": "Allow",
  "Principal": {"Service": "mediatailor.amazonaws.com"},
  "Action": "s3:GetObject",
  "Resource": "arn:aws:s3:::mybucket/*",
  "Condition": {
    "StringEquals": {"AWS:SourceAccount": "111122223333"}
  }
}
```

Example origin access policy for Amazon S3, scoped to the MediaTailor playback configuration

```
{
```



```

    "Effect": "Allow",
    "Principal": {"Service": "mediatailor.amazonaws.com"},
    "Action": "s3:GetObject",
    "Resource": "arn:aws:s3:::mybucket/*",
    "Condition": {
        "StringEquals": {"AWS:SourceArn": "arn:aws:mediatailor:us-
west-2:111122223333:playbackConfiguration/test"}
    }
}

```

MediaPackage requirements

If you use SigV4 to protect your MediaPackage v2 origin, the following requirements must be met for MediaTailor to access the manifest:

- The origin base URL in your MediaTailor configuration must be a MediaPackage v2 endpoint in the following format: `mediapackagev2.region.amazonaws.com`
- Your origin must be configured to use HTTPS. MediaTailor only supports HTTPS communication and does not allow HTTP connections. If HTTPS is not enabled at the origin, MediaTailor will not sign the request.
- Your channel must have an origin access policy that includes the following:
 - Principal access for MediaTailor to access your endpoint. Grant access to **mediatailor.amazonaws.com**.
 - IAM permissions **mediapackagev2:GetObject** to read all multivariant playlists referenced by the MediaTailor configuration.

For general information about SigV4 for MediaPackage v2, see the [Authenticating Requests \(AWS Signature Version 4\)](#) topic in the *MediaPackage v2 API reference*.

Example origin access policy for MediaPackage v2, scoped to the MediaTailor account

```

{
    "Effect": "Allow",
    "Principal": {"Service": "mediatailor.amazonaws.com"},
    "Action": "mediapackagev2:GetObject",
    "Resource": "arn:aws:mediapackagev2:us-west-2:444455556666:channelGroup/emp-origin-
channel-group/channel/emp-origin-channel/originEndpoint/emp-origin-endpoint",
    "Condition": {
        "StringEquals": {"AWS:SourceAccount": "444455556666"}
    }
}

```

```
}  
}
```

Example origin access policy for MediaPackage v2, scoped to the MediaTailor playback configuration

```
{  
  "Effect": "Allow",  
  "Principal": {"Service": "mediatailor.amazonaws.com"},  
  "Action": "mediapackagev2:GetObject",  
  "Resource": "arn:aws:mediapackagev2:us-west-2:444455556666:channelGroup/emp-origin-channel-group/channel/emp-origin-channel/originEndpoint/emp-origin-endpoint",  
  "Condition": {  
    "StringEquals": {"AWS:SourceArn": "arn:aws:mediatailor:us-west-2:444455556666:playbackConfiguration/test"}  
  }  
}
```

Integrating AWS Elemental MediaTailor with Google's ad platforms

MediaTailor automatically authenticates server-to-server connections with Google Ad Manager (GAM), Google Campaign Manager (GCM), and Google Display & Video 360 (DV360). When MediaTailor detects that an ad request or tracking event is destined for a Google ad server, it automatically establishes the required secure, authenticated connection.

You must have an account set up with the relevant Google ad platform. MediaTailor supports the following integration types:

- **Server-side integration:** MediaTailor automatically secures server-side ad requests and impression tracking requests to Google's ad platforms. This includes:
 - **Google Ad Manager (GAM):** Ad requests to Google's ad server for publishers are automatically secured, which is required for access to Authorized Buyers—Google's real-time ad sales marketplace and ad exchange.
 - **Google Campaign Manager (GCM):** Ad-serving and tracking requests are automatically routed through Google's authenticated endpoint and secured, supporting advertisers who run campaigns on these platforms with more accurate reporting and fewer rejected impressions.

- **Display & Video 360 (DV360):** Ad requests are automatically routed through Google's authenticated endpoint and secured, supporting advertisers who run campaigns on these platforms with more accurate reporting and fewer rejected impressions.
- **Client-side integration:** A player integration using the Google Programmatic Access Libraries (PAL) SDK. This integration is required if you want to use the open auction transaction type in Google Ad Manager.

All other ad requests continue to operate without modification.

Ad Manager support for programmatic transaction types varies based on the type of integration that you're using. For a list of available options, see [Transaction types](#) or contact your Google account team.

Topics

- [Server-side AWS Elemental MediaTailor integration with Google's ad platforms](#)
- [Client-side AWS Elemental MediaTailor integration with Google Ad Manager](#)
- [Optimizing ad fill rate with Google Ad Manager](#)

Server-side AWS Elemental MediaTailor integration with Google's ad platforms

MediaTailor automatically authenticates server-to-server connections with Google Ad Manager (GAM), Google Campaign Manager (GCM), and Google Display & Video 360 (DV360) using SSL certificates issued by Google to AWS Elemental MediaTailor. When MediaTailor detects ad requests or tracking events destined for Google ad servers, it automatically establishes the required secure, authenticated connection—no additional SSL certificate setup or support ticket is required.

Prerequisite: You must have an active account with the relevant Google ad platform (Google Ad Manager, Google Campaign Manager, or Display & Video 360).

MediaTailor applies Google-issued SSL certificate authentication based on the following destination URI rules:

Google Ad Manager (GAM): server-side ad requests

- This rule triggers on ad requests to `serverside.doubleclick.net`, including when you configure the ADS Template URL with that domain or when a redirect ad request resolves to it.
- MediaTailor adds the `ssss=mediatailor` query parameter if it is not present, then secures the connection using a Google-issued SSL certificate.
- This provides access to Authorized Buyers—Google's real-time ad sales marketplace and ad exchange.

Google Ad Manager (GAM): redirected ad requests inside VMAP

- This rule triggers on `pubads.g.doubleclick.net` redirected ad requests when MediaTailor has previously seen VAST redirects to `serverside.doubleclick.net` in the current request and a previous ADS response was a VMAP response.
- MediaTailor automatically changes the domain to `serverside.doubleclick.net` and establishes an authenticated connection using the Google-issued SSL certificate.
- This enables access to Authorized Buyers for VMAP-based ad workflows.

Google Campaign Manager (GCM): ad-serving and tracking requests

- This rule triggers on `ad.doubleclick.net` ad-serving and tracking requests when the URI contains `/ddm/pfadx/` or `/ddm/trackimp/` (encoded or unencoded).
- MediaTailor automatically changes the domain to `serverside.doubleclick.net` and establishes an authenticated connection using the Google-issued SSL certificate.
- This supports advertisers running campaigns on these platforms with more accurate reporting and fewer rejected impressions.

Display & Video 360 (DV360): ad requests

- This rule triggers on `bid.g.doubleclick.net` or `pagead2.googlesyndication.com` ad requests when the URI contains `/dbm/vast/` (encoded or unencoded).
- MediaTailor automatically changes the domain to `serverside.doubleclick.net` and establishes an authenticated connection using the Google-issued SSL certificate.
- This enables authenticated ad serving for Display & Video 360 campaigns.

All other ad requests are processed without Google-issued SSL certificate authentication.

Note

SSL certificate authentication is now automatic. If you previously submitted a support ticket to enable SSL certificates, no action is needed—your configurations continue to work.

Client-side AWS Elemental MediaTailor integration with Google Ad Manager

A MediaTailor client-side integration is required to use the Google Ad Manager Programmatic Access Libraries (PAL) SDKs and to use Google Ad Manager's open auction transaction type.

The PAL SDKs provide information about the content, device, and user data for a playback session. Through the PAL SDK, you can provide this information to Google Ad Manager, which can then make better determinations of what targeted ads to show. SDKs are available for Android, iOS, HTML5, and Cast. For information about using the PAL SDKs, see [Google Ad Manager PAL SDK](#).

To create client-side integration with Ad Manager

1. Use the PAL SDK to generate a nonce. For information about setting up a nonce, see your SDK documentation at [Google Ad Manager PAL SDK](#).

Note

The nonce is an encrypted string that PAL generates for stream requests. Each request must have a unique nonce.

2. Use the `givn` parameter in your ADS request to pass through the nonce value. To do this, update your ADS URL to include `&givn=[player_params.givn]`. For instructions, see [Enabling client-side tracking](#).

Datazoom player SDKs

MediaTailor has partnered with Datazoom to provide free player SDKs to ease integrations with SDKs such as those offered in the Google Ad Manager PAL. For information about the Datazoom and MediaTailor partnership, see [Datazoom free player SDKs](#). To access the Datazoom player SDKs, use the contact information on the [Datazoom with AWS](#) site.

Optimizing ad fill rate with Google Ad Manager

When you use Google Ad Manager (GAM) as your ad server with MediaTailor, your ADS template URL parameters directly affect ad fill rate and revenue. Incorrect parameter values are one of the most common causes of low fill rates.

Recommended GAM parameters

Include the following parameters in your ADS template URL to ensure GAM returns ads consistently and fills ad breaks to their full duration.

`correlator=[avail.random]`

Ensures GAM treats each ad break as a unique request and returns fresh ads. Without this parameter, GAM might return empty responses for repeated ad breaks within the same session.

`pmxd=[session.avail_duration_ms]`

The maximum ad pod duration in milliseconds. Use the MediaTailor dynamic variable `[session.avail_duration_ms]` so this value automatically matches the actual SCTE ad marker duration. Do not hardcode this value—if the hardcoded duration is shorter than the actual ad break, the remaining time goes unfilled.

`pmad=-1`

The maximum number of ads per pod. Set this to `-1` to allow as many ads as available to fill the break, or set it to your desired limit (such as `10`). A low value (such as `4`) artificially limits the number of ads that can fill a break, even when more inventory is available.

`vpos`

The video position (preroll, midroll, or postroll). Set this to the value that matches your actual ad slot type. Do not use `[avail.type]`—MediaTailor does not support this variable and it will resolve to an empty value, which can cause GAM to reject the request or return low-value inventory.

Player parameters for GAM

Some GAM parameters require values that the publisher or player application provides at session initialization. Pass these to MediaTailor as player parameters and reference them in your ADS template URL. For details on configuring player parameters, see [MediaTailor player variables for ADS requests](#).

`ppid=[player_params.ppid]`

The Programmatic Publisher-Provided Identifier. Provide this publisher-assigned value so that GAM can track sessions and continue returning ads with each ad break. Without this, GAM might stop returning ads after the first break in a session.

`rdid=[player_params.rdid]` (Resettable Device Identifier)

The device advertising ID. For CTV and Smart TV devices, programmatic ad buyers require this identifier to bid on inventory. Without it, fill rates on CTV devices can be significantly lower.

`msid=[player_params.msid]` (App ID)

The application identifier. Like `rdid`, this is required by programmatic buyers for CTV and Smart TV device targeting.

For more information about device targeting parameters that GAM supports, see [Google Ad Manager targeting parameters](#) on the Google website.

Common configuration mistakes

The following configuration mistakes are common causes of low fill rates with GAM:

- **Hardcoded `pmxd`**—Using a fixed value (such as `pmxd=60000`) when actual ad breaks are longer (such as 90 seconds) leaves the remaining time unfilled. Use `[session.avail_duration_ms]` instead.
- **Low `pmad` value**—Setting the maximum ads per pod too low prevents GAM from filling the full break duration even when inventory is available. Use `-1` for unlimited, or set your desired limit.
- **Missing `correlator`**—Without this parameter, GAM might return empty VAST responses for repeated ad breaks within the same session.
- **Missing `ppid`**—Without this player parameter, GAM might stop returning ads after the first ad break in a session.
- **Missing device ID on CTV devices**—Without `rdid`, programmatic buyers cannot target the device, which significantly reduces fill for CTV inventory.
- **Using `vpos=[avail.type]`**—MediaTailor does not support the `[avail.type]` variable. This resolves to an empty string, which can cause request failures or low-value ad responses.
- **Player parameter not passed at session initialization**—Your ADS template URL references a player parameter (such as `[player_params.ppid]`), but the session initialization call does not

pass the corresponding value. GAM receives an empty or null value for that macro, which can cause the ad request to return no fill.

Customizing ad break behavior with ad break suppression

When you create a configuration in AWS Elemental MediaTailor, you can specify optional ad break configuration settings that govern the behavior of ad breaks, including the ability to configure ad break suppression. This allows you to tailor the ad break experiences for your video content to meet your specific requirements.

Compatibility restrictions

You can't use ad break suppression with the following:

- VOD and live-to-VOD workflows. Only live workflows are supported.
- Server-guided ad insertion (SGAI) methods. Server-guided methods handle ad decisioning differently and don't require suppression configuration.

Topics

- [Configuring ad break suppression](#)

Configuring ad break suppression

You can configure MediaTailor to skip ad break personalization for live content. This is known as *ad break suppression*, or *avail suppression*. This topic shows you how, and it also explains how configuring ad break suppression works.

Ad break suppression can be used for the following use cases:

- **Large manifest lookback window** – If a viewer starts playback at the live edge of a manifest but the lookback window is large, you might want to only insert ads starting after the viewer started watching. Or, insert ads for a portion of the total lookback window in the manifest. You can configure ad suppression so that MediaTailor personalizes ad breaks on or within a specified time range behind the live edge.
- **Mid-break join** – If the viewer starts watching a live video stream in the middle of an ad break, that user is likely to change the channel and not watch the advertisement. With ad suppression, you can skip ad break personalization if the ad break started before the viewer joined the stream.

Configuring ad suppression

To use ad suppression, you configure an **avail suppression mode**, **avail suppression value**, and **avail suppression fill policy** in the following ways:

- In the MediaTailor console
- Using the AWS Command Line Interface (AWS CLI)
- Using MediaTailor API, or as parameters in your client's playback session request

For information about configuration with parameters, see [Configuring ad suppression parameters – playback session request](#).

Ad suppression configuration parameters

You can choose to turn on or turn off ad suppression. If you turn on ad suppression, you specify whether that suppression happens after the live playback edge or before the live playback edge of a live stream. In either case, you also specify a time, relative to the live edge, where MediaTailor doesn't personalize ads. When you turn on avail suppression, you can specify an avail suppression policy that MediaTailor uses for partial ad-break fills when a session starts mid-break.

The following are the ad suppression configuration parameters:

- **Avail suppression mode** – Sets the ad suppression mode. By default, ad suppression is off.
Accepted values: OFF, BEHIND_LIVE_EDGE, or AFTER_LIVE_EDGE.
 - OFF: There is no ad suppression and MediaTailor personalizes all ad breaks.
 - BEHIND_LIVE_EDGE: MediaTailor doesn't personalize ad breaks that start before the live edge, minus the **Avail suppression value**. This affects the entire ad break, not just individual ad avails.
 - AFTER_LIVE_EDGE: MediaTailor doesn't personalize ad breaks that are within the live edge, plus the **Avail suppression value**. This can be configured to affect either entire ad breaks or allow partial filling of ad avails.
- **Avail suppression value** – A time relative to the live edge in a live stream. **Accepted value:** A time value in HH:MM:SS.
- **Avail suppression fill policy** – Defines the policy that MediaTailor applies to the **Avail suppression mode**. **Accepted values:** PARTIAL_AVAIL, FULL_AVAIL_ONLY.
 - BEHIND_LIVE_EDGE mode always uses the FULL_AVAIL_ONLY suppression policy.

- AFTER_LIVE_EDGE mode can be used to invoke PARTIAL_AVAIL ad break fills when a session starts mid-break.

Ad suppression settings examples

The way in which the [ad suppression configuration parameters](#) interact with one another lets you specify several different ways to handle ad suppression and avail filling before, at, or after the live edge of the live stream. This section provides examples that show you some of these interactions. Use these examples to help you set up the configuration parameters for your particular situation.

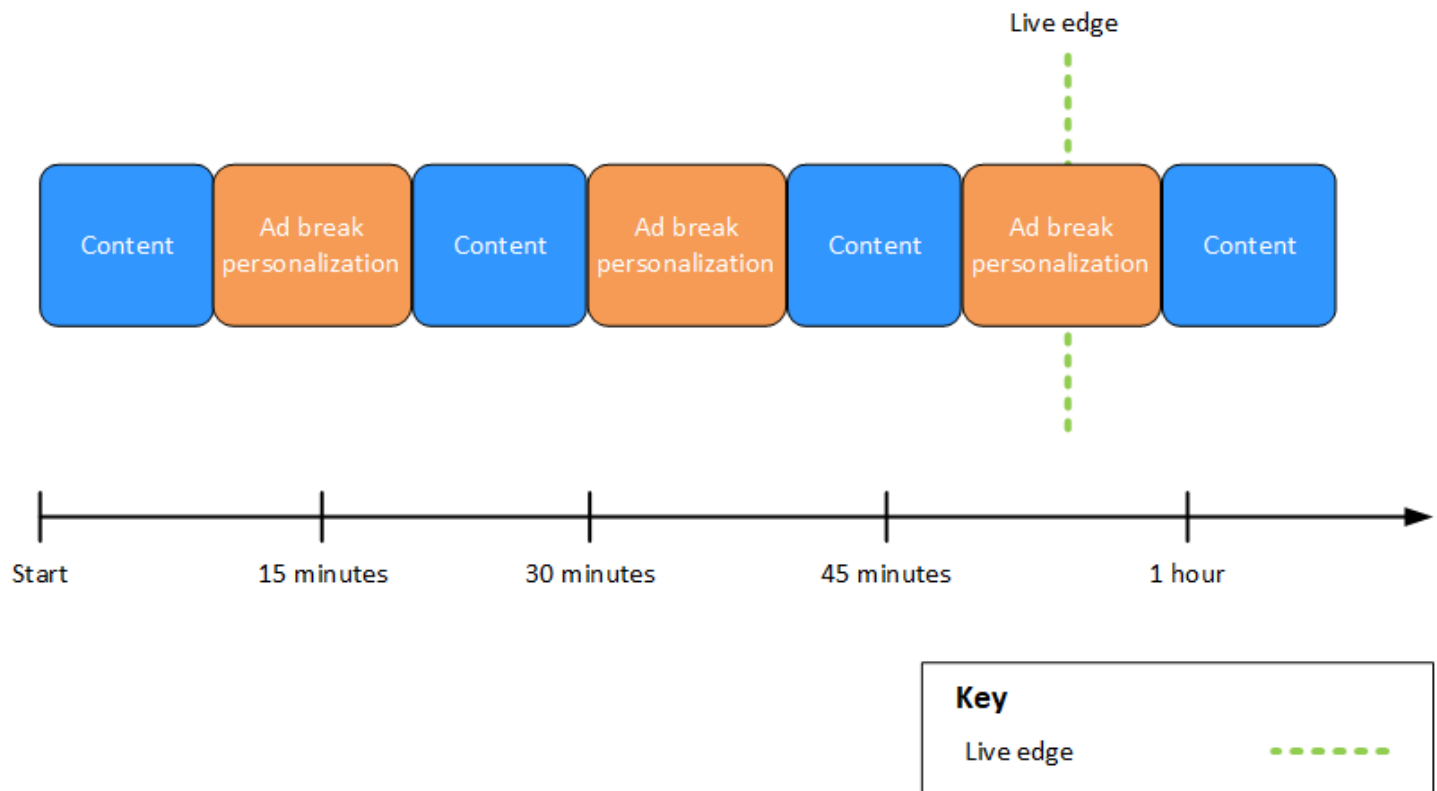
The following are examples of ad suppression settings:

Example 1: No ad suppression

When the **avail suppression mode** is OFF, there is no ad suppression and MediaTailor personalizes all ad breaks.

In the following figure, various blocks are arranged horizontally along a timeline that progresses from left to right. Each block represents a portion of time where the content of the live stream or a personalized ad break plays. A dotted line represents the current live edge of the live stream. Two ad breaks occur before the live edge, and another ad break is in progress at the live edge. As shown in the figure, when the avail suppression mode is OFF, MediaTailor personalizes all ad breaks that occur before the live edge on the timeline. MediaTailor also personalizes the ad break in progress at the live edge.

Avail suppression mode (default): OFF



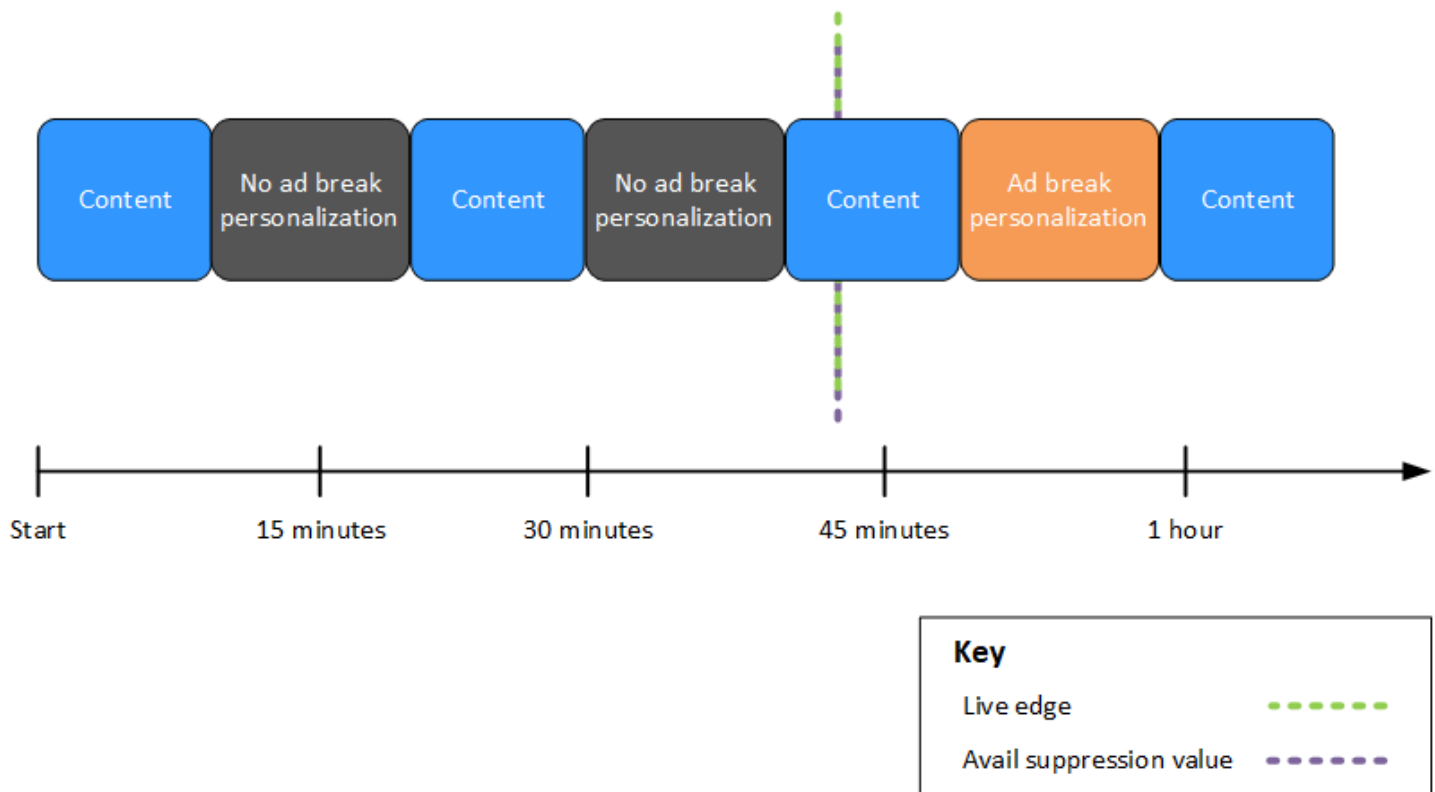
Example 2: BEHIND_LIVE_EDGE ad suppression with value in sync with live edge

When **avail suppression mode** is set to `BEHIND_LIVE_EDGE` and the **avail suppression value** is set to `00:00:00`, the avail suppression value is in sync with the live edge. MediaTailor doesn't personalize any ad breaks that start on or before the live edge.

In the following figure, various blocks are arranged horizontally along a timeline that progresses from left to right. Each block represents a portion of time where the content of the live stream, a personalized ad break, or a non-personalized ad break plays. A dotted line represents the current live edge of the live stream. Another dotted line, representing the avail suppression value set to `00:00:00`, overlaps the dotted line for the live edge. Two ad breaks occur before the live edge, and another ad break occurs after the live edge. As shown in the figure, when the avail suppression mode is set to `BEHIND_LIVE_EDGE`, and the avail suppression value is set to `00:00:00` so that it's in sync with the live edge, MediaTailor doesn't personalize any ad breaks that occur before the live edge on the timeline. MediaTailor personalizes the ad break that occurs *after* the live edge.

Avail suppression mode: **BEHIND_LIVE_EDGE**

Avail suppression value: **00:00:00**



Example 3: BEHIND_LIVE_EDGE ad suppression with value behind live edge

When the **avail suppression mode** is set to **BEHIND_LIVE_EDGE**, MediaTailor doesn't personalize any ad breaks on or before that time. In this example, MediaTailor personalizes ad breaks that start up to 45 minutes behind the live edge. MediaTailor *doesn't* personalize ad breaks that start more than 45 minutes behind the live edge.

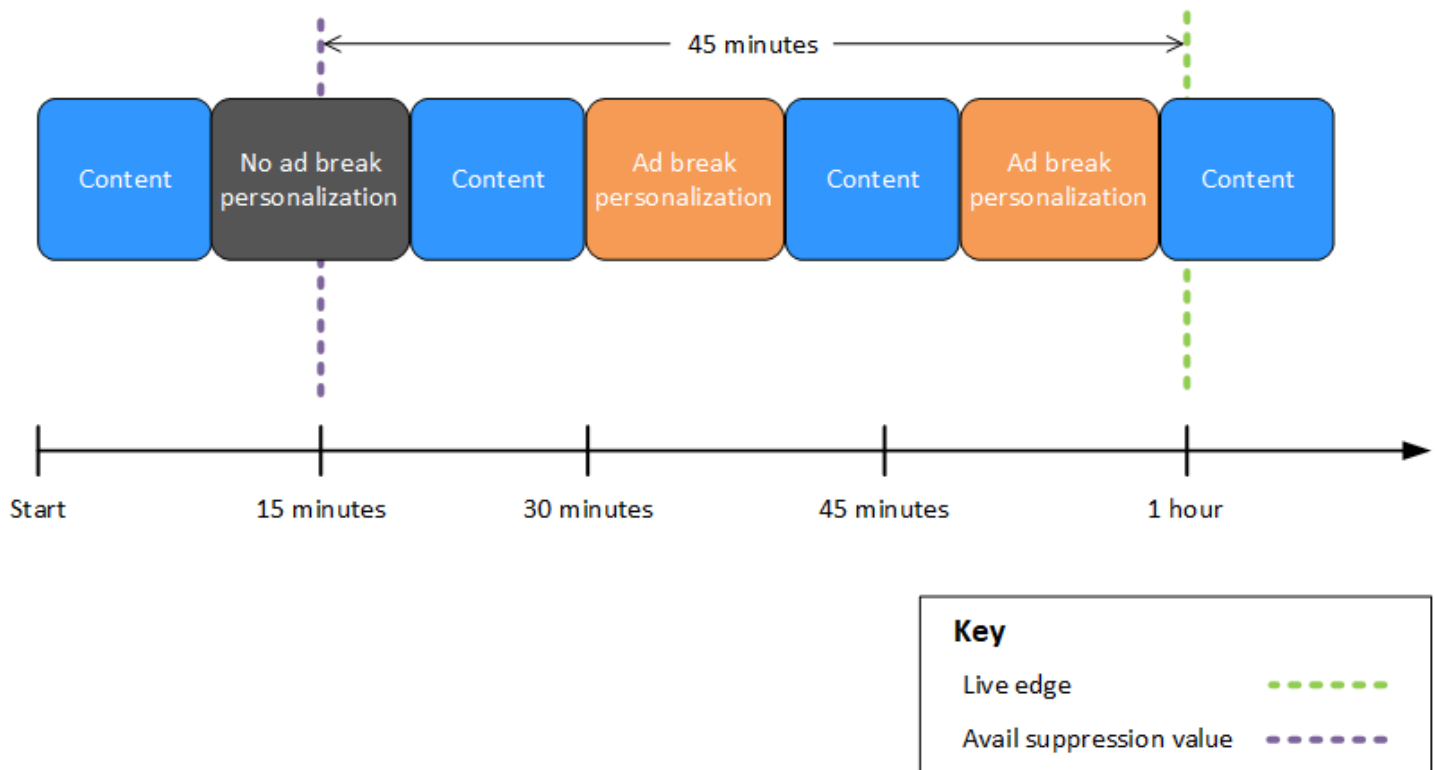
In the following figure, various blocks are arranged horizontally along a timeline that progresses from left to right. Each block represents a portion of time where the content of the live stream, a personalized ad break, or a non-personalized ad break plays. A dotted line represents the current live edge of the live stream. Another dotted line, representing the avail suppression value set to 00:45:00, occurs 45 minutes earlier in the timeline with respect to the dotted line for the live edge. The 45-minute time period between the dotted lines represents the avail suppression period. An ad break is in progress at the beginning of the avail suppression period. Two other ad breaks occur during the avail suppression period.

As shown in the figure, when the avail suppression mode is set to **BEHIND_LIVE_EDGE**,

and the avail suppression value is set to 00:45:00 behind the live edge, MediaTailor personalizes any ad breaks that occur within the avail suppression period. MediaTailor *doesn't* personalize the ad break in progress at the beginning of the avail suppression period.

Avail suppression mode: **BEHIND_LIVE_EDGE**

Avail suppression value: 00:45:00



Example 4: AFTER_LIVE_EDGE ad suppression with no ad breaks occurring during the avail suppression period

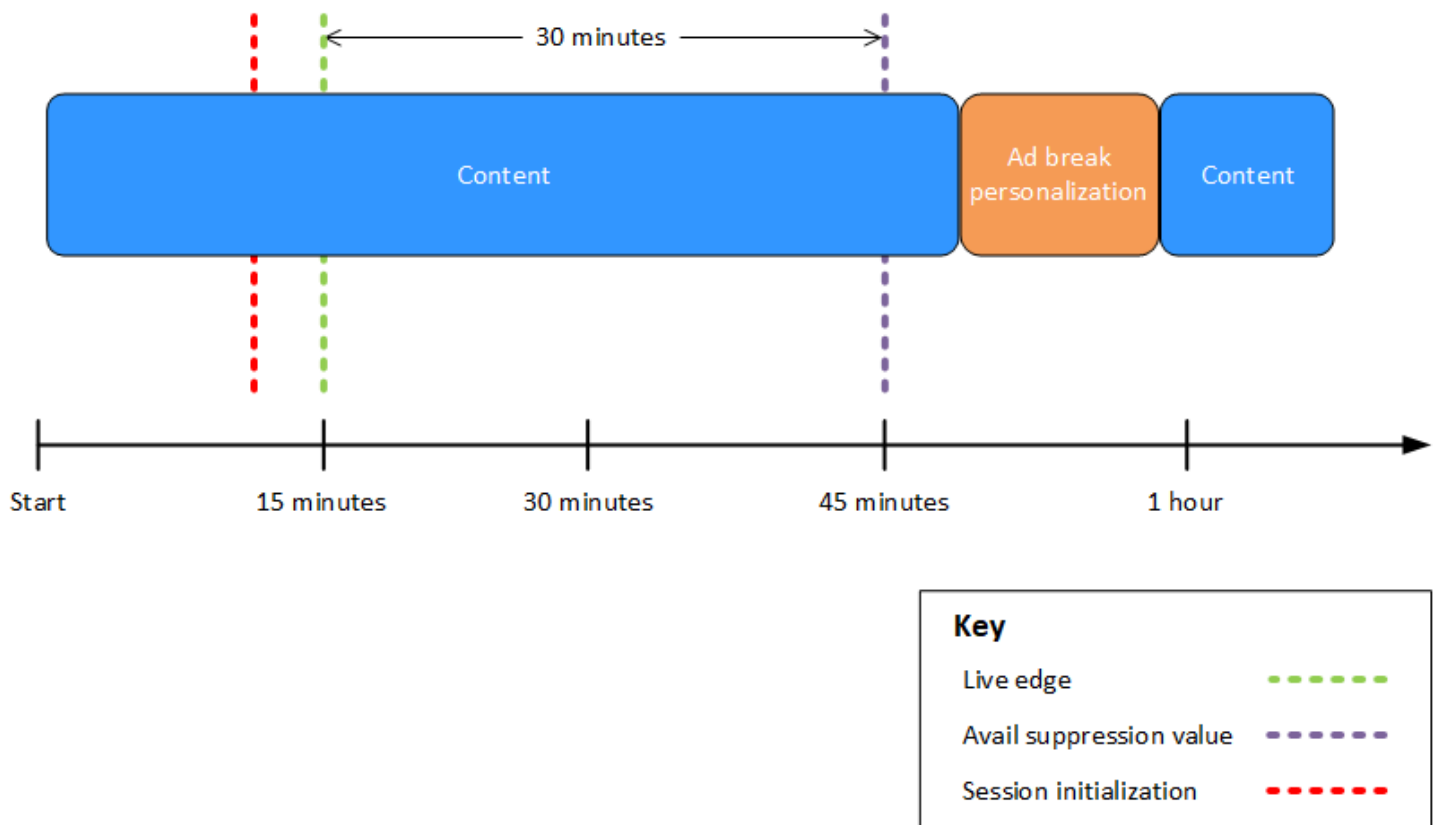
When the **avail suppression mode** is set to **AFTER_LIVE_EDGE** and the **avail suppression value** is greater than zero, MediaTailor doesn't personalize any ad breaks until the elapsed time of the session has reached that value.

In the following figure, various blocks are arranged horizontally along a timeline that progresses from left to right. Each block represents a portion of time where the content of the live stream or a personalized ad break plays. A dotted line represents the current live edge of the live stream. Another dotted line, representing the avail suppression value set to 00:30:00, occurs 30 minutes later in the timeline with respect to the dotted line for the live edge. A third dotted line, representing the session initialization, occurs earlier in the

timeline with respect to the dotted line for the live edge. The 30-minute time period between the live-edge time and the avail-suppression-value time represents the avail suppression period. An ad break occurs after the avail suppression period. As shown in the figure, when the avail suppression mode is set to `AFTER_LIVE_EDGE`, the avail suppression value is set to `00:30:00` after the live edge, and the session initialization occurs before the live edge, MediaTailor personalizes any ad breaks that occur *after* the avail suppression period.

Avail suppression mode: `AFTER_LIVE_EDGE`

Avail suppression value: `00:30:00`



Example 5: `AFTER_LIVE_EDGE` ad suppression with `PARTIAL_AVAIL` fill policy and an ad break in progress at the end of the avail suppression period

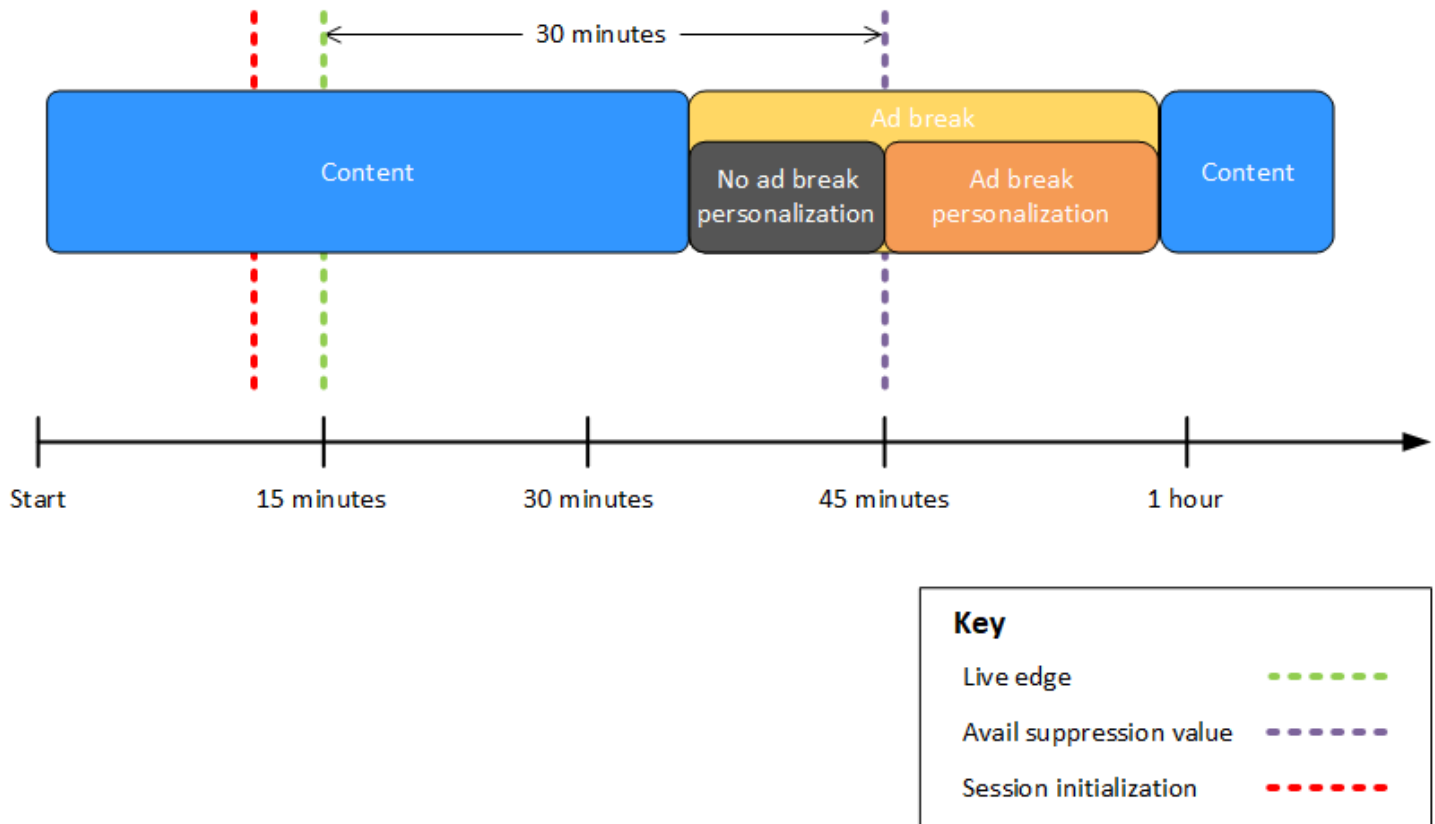
When the **avail suppression mode** is set to `AFTER_LIVE_EDGE` and the **avail suppression value** is greater than zero, MediaTailor doesn't personalize any ad breaks until the elapsed time of the session has reached that value.

In the following figure, various blocks are arranged horizontally along a timeline that progresses from left to right. Each block represents a portion of time where the content of the live stream, a personalized ad break, or a non-personalized ad break plays. A dotted line represents the current live edge of the live stream. Another dotted line, representing the avail suppression value set to 00:30:00, occurs 30 minutes later in the timeline with respect to the dotted line for the live edge. A third dotted line, representing the session initialization, occurs earlier in the timeline with respect to the dotted line for the live edge. The 30-minute time period between the live-edge time and the avail-suppression-value time represents the avail suppression period. An ad break is in progress at the end of the avail suppression period. As shown in the figure, when the avail suppression mode is set to AFTER_LIVE_EDGE, the avail suppression value is set to 00:30:00 after the live edge, the avail suppression fill policy is set to PARTIAL_AVAIL, and the session initialization occurs before the live edge, MediaTailor personalizes any ad breaks that occur *after* the avail suppression period. For the ad break in progress at the end of the avail suppression period, MediaTailor personalizes the portion of that ad break which occurs *after* the avail suppression period, but doesn't personalize the portion of that ad break which occurs *during* the avail suppression period.

Avail suppression mode: `AFTER_LIVE_EDGE`

Avail suppression value: `00:30:00`

Avail suppression fill policy: `PARTIAL_AVAIL`



Example 6: `AFTER_LIVE_EDGE` ad suppression with `PARTIAL_AVAIL` fill policy and an ad break in progress from before session initialization to after the end of the avail suppression period

When the **avail suppression mode** is set to `AFTER_LIVE_EDGE` and the **avail suppression value** is greater than zero, MediaTailor doesn't personalize any ad breaks until the elapsed time of the session has reached that value.

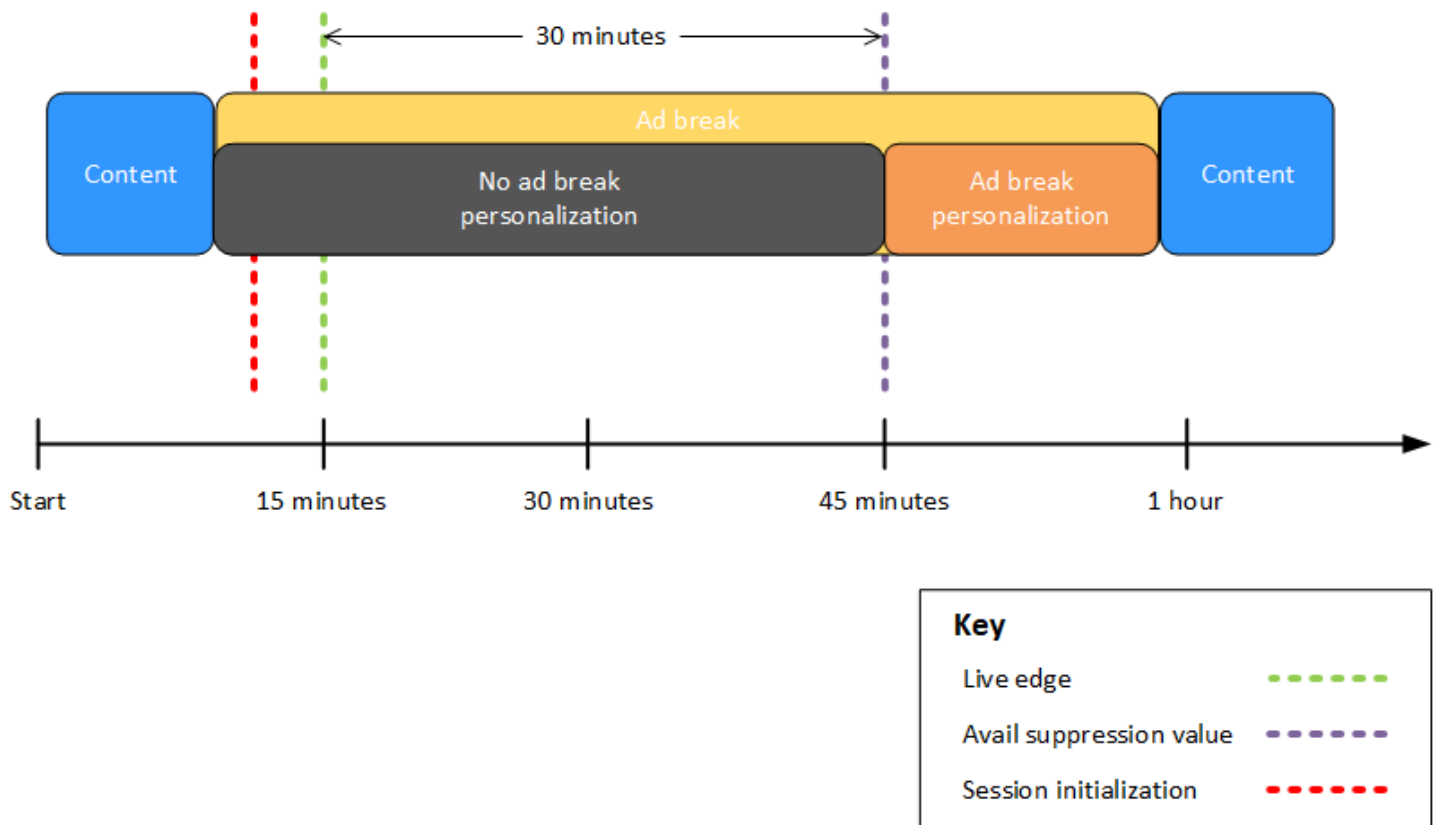
In the following figure, various blocks are arranged horizontally along a timeline that progresses from left to right. Each block represents a portion of time where the content of the live stream, a personalized ad break, or a non-personalized ad break plays. A dotted line represents the current live edge of the live stream. Another dotted line, representing the avail suppression value set to `00:30:00`, occurs 30 minutes later in the timeline with respect to the dotted line for the live edge. A third dotted line, representing the session initialization, occurs earlier in the timeline with respect to the dotted line for the live edge. The 30-minute time period between the live-

edge time and the avail-suppression-value time represents the avail suppression period. An ad break is in progress from a time before session initialization to a time after the avail suppression period. As shown in the figure, when the avail suppression mode is set to `AFTER_LIVE_EDGE`, the avail suppression value is set to `00:30:00` after the live edge, the avail suppression fill policy is set to `PARTIAL_AVAIL`, and the session initialization occurs before the live edge, MediaTailor personalizes any ad breaks that occur *after* the avail suppression period. For the ad break in progress before, during, and after the avail suppression period, MediaTailor personalizes the portion of that ad break which occurs *after* the avail suppression period, but doesn't personalize the portion of that ad break which occurs *before* or *during* the avail suppression period.

Avail suppression mode: `AFTER_LIVE_EDGE`

Avail suppression value: `00:30:00`

Avail suppression fill policy: `PARTIAL_AVAIL`



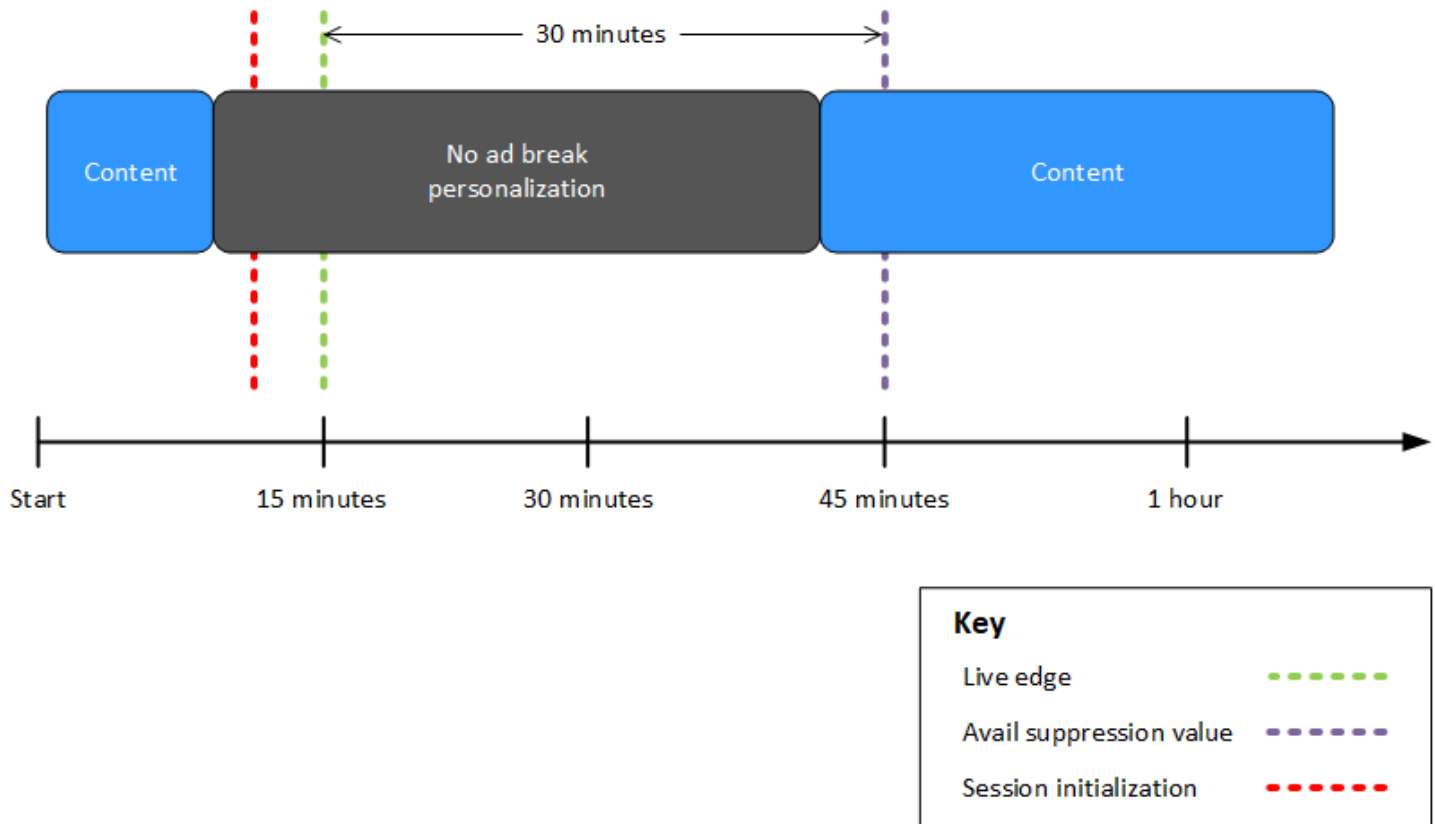
Example 7: AFTER_LIVE_EDGE ad suppression with an ad break in progress at the beginning of the avail suppression period

When the **avail suppression mode** is set to AFTER_LIVE_EDGE and the **avail suppression value** is greater than zero, MediaTailor doesn't personalize any ad breaks until the elapsed time of the session has reached that value.

In the following figure, various blocks are arranged horizontally along a timeline that progresses from left to right. Each block represents a portion of time where the content of the live stream or a non-personalized ad break plays. A dotted line represents the current live edge of the live stream. Another dotted line, representing the avail suppression value set to 00:30:00, occurs 30 minutes later in the timeline with respect to the dotted line for the live edge. A third dotted line, representing the session initialization, occurs earlier in the timeline with respect to the dotted line for the live edge. The 30-minute time period between the live-edge time and the avail-suppression-value time represents the avail suppression period. An ad break is in progress from a time before session initialization to a time within the avail suppression period. As shown in the figure, when the avail suppression mode is set to AFTER_LIVE_EDGE, the avail suppression value is set to 00:30:00 after the live edge, and the session initialization occurs before the live-edge time but after the start of the ad break, MediaTailor doesn't personalize that ad break.

Avail suppression mode: **AFTER_LIVE_EDGE**

Avail suppression value: **00:30:00**



Configuring ad suppression parameters – playback session request

You can configure ad suppression settings via parameters in your *initial* server-side or client-side playback session request to MediaTailor. If you've already configured ad suppression settings via the MediaTailor Console or AWS Elemental MediaTailor API, these parameters override those settings.

Both the avail suppression mode and avail suppression value are required for ad suppression to work. These parameters can't be configured from different sources. For example, you can't configure one parameter with the MediaTailor console and another with a query parameter.

MediaTailor supports the following ad suppression parameters.

Name	Description	Accepted Values
<code>availSuppressionMode</code>	Sets the mode for ad suppression. By default, ad suppression is OFF. When set to <code>BEHIND_LIVE_EDGE</code> , MediaTailor doesn't fill ad breaks on or behind the <code>aws.availSuppressionValue</code> time. When set to <code>AFTER_LIVE_EDGE</code> , MediaTailor doesn't fill ad breaks on or behind the avail suppression period. The avail suppression period spans from the live-edge time to the <code>aws.availSuppressionValue</code> time, plus additional buffer time.	- OFF <code>BEHIND_LIVE_EDGE</code> <code>AFTER_LIVE_EDGE</code>
<code>availSuppressionValue</code>	A time relative to the live edge in a live stream.	A UTF-8 URL-encoded time code in HH:MM:SS. For example, 1 hour and 30 minutes would be <code>01%3A30%3A00</code> .
<code>availSuppressionFillPolicy</code>	Defines the policy to apply to the avail suppression mode. <code>BEHIND_LIVE_EDGE</code> always uses the full avail suppression policy. <code>AFTER_LIVE_EDGE</code> can be used to invoke partial ad break fills when a session starts mid-break.	<code>PARTIAL_AVAILABLE</code> - not available for the <code>BEFORE_LIVE_EDGE</code> suppression mode <code>FULL_AVAILABLE_ONLY</code> - the default value for the <code>AFTER_LIVE_EDGE</code> suppression mode

Server-side configuration

The base query parameter is `aws.availSuppression`, which is followed by optional parameter name and value pairs. To construct the query, append `aws.availSuppression=` to the end of the playback session request to MediaTailor, followed by parameter names and values. For more information about how to construct a server-side playback session request, see [MediaTailor server-side ad tracking and reporting](#).

Example: HLS

```
GET <mediatailorURL>/v1/master/<hashed-account-id>/<origin-id>/index.m3u8?
aws.availSuppressionMode=BEHIND_LIVE_EDGE&aws.availSuppressionValue=00%3A00%3A21
```

Server-side query syntax is listed in the following table.

Query String Component	Description
?	A restricted character that marks the beginning of a query.
aws .	The base query, which is followed by parameters constructed of name and value pairs. For a list of all of the available parameters, see Configuring ad suppression parameters – playback session request .
=	Associates the parameter name with a value. For example, <code>aws.availSuppressionMode= BEHIND_LIVE_EDGE</code> .
&	Concatenates query parameters. For example, <code>aws.availSuppressionMode= BEHIND_LIVE_EDGE &aws.availSuppressionValue= 00:30:00&aws.availSuppressionFillPolicy= FULL_AVAIL_ONLY ></code> .

Client-side configuration

Include `availSuppression` parameters in your client's POST request to MediaTailor. For more information about how to construct a client-side playback session request, see [Client-side ad tracking](#).

Example: HLS

```
POST parent.m3u8
{
  "availSuppression": {
    "mode": "BEHIND_LIVE_EDGE",
    "value": "00:00:21",
    "fillPolicy": "FULL_AVAIL_ONLY"
  }
}
```

MediaTailor bumper ad insertion

Bumpers are short, non-skippable video or audio clips that play at the start or before the end of an ad break with AWS Elemental MediaTailor.

The following conditions apply to bumpers:

- Bumpers must be 10 seconds or less.
- Bumpers can be inserted at the start of an ad break, directly before the end of an ad break, or both.
- Bumpers play during every ad break in a playback session unless pre-roll is configured. If pre-roll is configured, bumpers won't play during the pre-roll break. Instead, they will play in every subsequent break after the pre-roll.
- For HLS, you must include the `duration` attribute with each SCTE-35 EXT-X-CUE-OUT tag.
- Bumpers are transcoded to match source content.
- You are not charged for bumpers.

Configuring bumpers

To use bumpers, configure the bumper URLs with the MediaTailor console, MediaTailor API, or the AWS Command Line Interface (AWS CLI). You can configure a start bumper, an end bumper, or both. Bumpers are stored on a server, such as Amazon Simple Storage Service (Amazon S3). The bumper URLs indicate the location of the stored bumper asset(s).

Example start and end bumper URLs:

Start bumper URL: `https://s3.amazonaws.com/startbumperad`

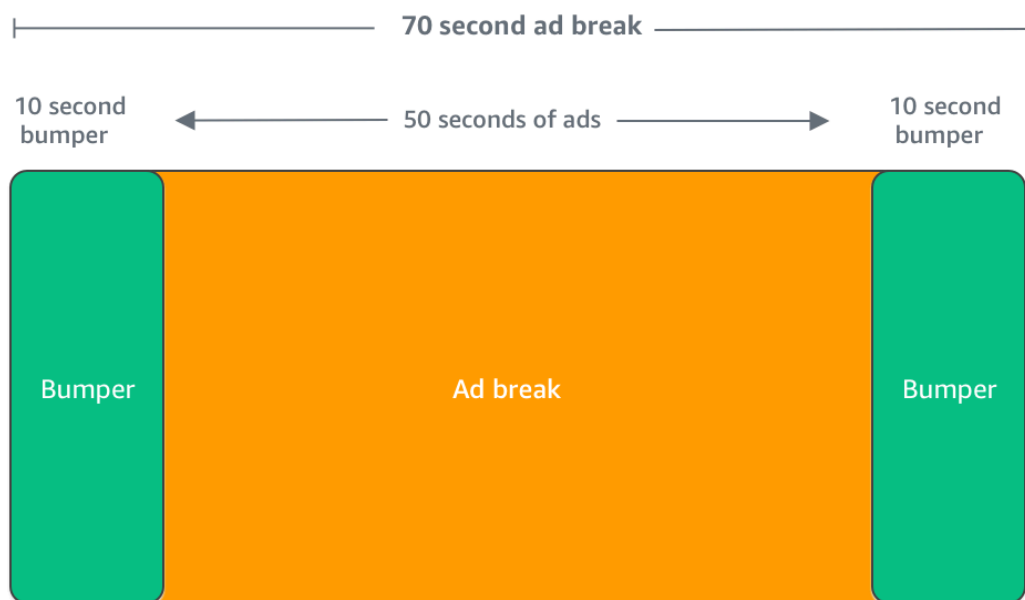
End bumper URL: `https://s3.amazonaws.com/endbumperad`

Example

The following is an example of bumper ad behavior.

Example 1: Start and end bumpers

In this example, start and end bumpers are enabled. The ad decision server has 50 seconds of personalized ads to fill a 70-second ad break. The 10-second start bumper plays at the start of the ad break, 50 seconds of ads plays, then the 10-second end bumper.



MediaTailor pre-roll ad insertion

Note

Configurable pre-roll ads are only available for live workflows. For details about how ad insertion (including pre roll) works for VOD, see [Ad stitching behavior for VOD](#).

AWS Elemental MediaTailor can insert ads at the beginning of a playback session, before the main content begins. These are *pre-roll* ads.

To insert pre-roll ads, complete the **Live pre-roll ad decision server** and **Live pre-roll maximum allowed duration** fields in the **Additional** settings on your configuration, as described in [Optional configuration settings](#).

1. When MediaTailor receives a playback request, it sends a request to the ADS for pre-roll ads based on the following fields in the MediaTailor playback configuration:
 - **Live pre-roll ad decision server** is the ad decision server (ADS) URL where MediaTailor sends the request for pre-roll ads.
 - **Live pre-roll maximum allowed duration** is the total maximum length of time for the pre-roll ads. MediaTailor takes the following action based on the maximum allowed duration:
 - If the total duration of the ads in the ADS response is *less* than the value you gave in **Live pre-roll maximum allowed duration**, MediaTailor inserts all of the ads. When the last ad is complete, MediaTailor immediately returns to the underlying content.
 - If the total duration of the ads in the ADS response is *more* than the value you gave in **Live pre-roll maximum allowed duration**, MediaTailor selects a set of ads that fit into the duration without going over. MediaTailor inserts these ads without clipping or truncation. MediaTailor returns to the underlying content when the last selected ad completes.
2. When MediaTailor receives the pre-roll response from the ADS, it manipulates the manifest to add links to the pre-roll ads. MediaTailor calculates the start time of the pre-roll ad break as follows:
 - For DASH, the formula is $(\text{publishTime} - \text{availabilityStartTime}) - \max(\text{suggestedPresentationDelay}, \text{minBufferTime})$.
 - For HLS, the formula is $\max(2 * \text{EXT-X-TARGETDURATION}, \text{EXT-X-START:TIMEOFFSET})$.
3. MediaTailor determines what action to take on any ad breaks that aren't pre-rolls. If the pre-roll overlaps another ad break, MediaTailor doesn't personalize the overlapped portion of the ad break.

MediaTailor slate ad insertion

Note

Slate is only available for live and live-to-VOD workflows.

With AWS Elemental MediaTailor, you can designate a *slate ad* for ad breaks. A slate is a default MP4 asset that is inserted into a stream, such as a still image or looped video, that plays instead of the live content.

AWS Elemental MediaTailor shows slate during ad breaks in the following specific situations:

- To fill in time remaining in a break that has not been replaced by personalized ads
- If the Ad Decision Server (ADS) responds with an empty VAST or VMAP response
- For error conditions, such as ADS timeout or HTTP 500 error from the ADS
- If an ad isn't available for insertion by MediaTailor (such as when it hasn't finished transcoding)

If you don't configure a slate, MediaTailor defaults to the underlying content stream when one of the above conditions is met.

Configuring the slate

You designate the slate in the **additional configuration** pane in the [MediaTailor console](#). MediaTailor downloads the slate from the URL that you specify, and transcodes it to the same renditions as your content. You can control the maximum amount of time that a slate will be shown through the optional **personalization threshold** configuration in the MediaTailor console. For more information, see [the section called “How personalization threshold works”](#).

The slate must be a high-quality MP4 asset that contains both audio and video. Configuring the slate is optional for non-VPAID configurations but mandatory for VPAID workflows.

Note

If the server that hosts your slate uses HTTPS, its certificate must be from a well-known certificate authority. It can't be a self-signed certificate. If you use a self-signed certificate, then AWS Elemental MediaTailor can't retrieve and stitch the slate into the manifests from the content origin.

How personalization threshold works

The personalization threshold defines the maximum duration of underfilled ad time (in seconds) allowed in an ad break. This feature applies specifically to ad replacement in Live and VOD streams, rather than ad insertion, because it relies on an underlying content stream.

The behavior varies based on three scenarios:

1. When personalization is disabled:

- a. Slate will be inserted for the full duration of unfilled time
- b. Start/End bumpers will be inserted when configured (for more information, see [MediaTailor bumper ad insertion](#))
- c. Ads will be inserted as normal

2. When personalization is enabled and threshold less than break duration:

- a. If the unfilled time exceeds the personalization threshold:
 - i. MediaTailor abandons personalization of the ad break
 - ii. The underlying content is shown
 - iii. No ads, slate, or bumpers are inserted
- b. If the unfilled time is less than the personalization threshold:
 - i. Ads and slate are inserted
 - ii. Bumpers are inserted if configured

3. When personalization is enabled and threshold greater than break duration:

- a. Ads will be inserted
- b. Slate will be inserted for any remaining time in the ad break
- c. Bumpers will be inserted if configured

The following table provides detailed behaviors for specific scenarios across both HLS and DASH protocols:

Detailed behavior matrix

Scenario	Personalization disabled	Personalization enabled less than break duration	Personalization enabled greater than break duration
Empty VAST or VMAP	Slate inserted	No slates inserted	Slate inserted
ADS timeout	Slate inserted	No slates inserted	Slate inserted

Scenario	Personalization disabled	Personalization enabled less than break duration	Personalization enabled greater than break duration
Ad isn't available (Vast 404)	Slate inserted	No slates inserted	Slate inserted
Duration of ads is longer than ad break	Ads inserted	No ads inserted	Ads inserted
Fill in time that's not fully used by an ad replacement	Slate inserted	If personalization threshold greater than unfilled time: No ads/slides inserted, else: Ads and slates inserted	Slate inserted
Preroll with empty VAST	Slate inserted, no preroll inserted	No slates inserted, no preroll inserted	Slate inserted, no preroll inserted
Preroll with ADS timeout	Slate inserted, no preroll inserted	No slates inserted, no preroll inserted	Slate inserted, no preroll inserted
Preroll when ad isn't available (Vast 404)	Slate inserted, no preroll inserted	No slates inserted, no preroll inserted	Slate inserted, no preroll inserted
Bumpers with empty VAST	Slate inserted, bumper inserted	No slates inserted, no bumpers inserted	Slate inserted, bumper inserted

Scenario	Personalization disabled	Personalization enabled less than break duration	Personalization enabled greater than break duration
Bumpers with ADS timeout	Slate inserted, bumper inserted	No slates inserted, no bumpers inserted	Slate inserted, bumper inserted
Bumpers when ad isn't available (Vast 404)	Slate inserted, bumper inserted	No slates inserted, no bumpers inserted	Slate inserted, bumper inserted
Bumpers to fill in time that's not fully used by ad replacement	Slate inserted, bumper inserted, ad inserted	If personalization threshold greater than unfilled time: No bumpers/ads/slides inserted, else: No bumpers/ads/slides inserted	Slate inserted, bumper inserted, ad inserted

Important considerations

Remember the following when you're working with slate and personalization threshold.

- This behavior is consistent across both HLS and DASH protocols
- The personalization threshold feature only applies when slate is configured
- When slate is configured and personalization threshold is not configured, slate will play for either the full origin avail duration or whatever time remains after filling in the ads
- For VPAID ads, MediaTailor inserts slate for the duration of the VPAID ad to hold space for ads that the video player will insert

Slate configuration and VPAID

Important

Slate configuration is mandatory when using VPAID ads. MediaTailor inserts slate to hold space for VPAID ads that the video player will insert. The slate duration might be slightly longer than the VPAID ad duration to accommodate user interactivity.

The video player then handles the VPAID ad based on the client-side reporting metadata that MediaTailor returns, as described in [VPAID requirements](#). For information about client-side reporting, see [the section called “Client-side tracking”](#).

Ad transcoding in AWS Elemental MediaTailor

AWS Elemental MediaTailor transcodes ads, bumpers, and slate to match the format and specifications of your origin video streams, ensuring seamless playback during ad breaks. MediaTailor supports two transcoding approaches: dynamic transcoding (the default) and a custom transcode profile. Understanding the differences between these approaches will help you determine the right configuration for your workflows.

Topics

- [Dynamic transcoding](#)
- [Custom transcode profiles](#)

Dynamic transcoding

Dynamic transcoding is the default ad transcoding behavior in MediaTailor. When an ad, bumper, or slate needs to be transcoded, MediaTailor automatically inspects the origin manifest and derives transcoding settings at runtime, without any configuration on your part.

What dynamic transcoding handles

Dynamic transcoding automatically handles the following for HLS and DASH workflows.

- **Standard codec matching** – MediaTailor matches the codec profile of the origin stream. Note that codec level matching is not guaranteed.

- **Audio normalization** – MediaTailor automatically applies audio normalization to transcoded ads using a default target level of -24 LKFS. This ensures that ad audio levels are consistent with your origin content without any additional configuration. If your workflow requires a custom audio normalization target other than the default of -24 LKFS, a custom transcode profile is required.
- **Standard segment length matching** – By default, MediaTailor transcodes ads using 2-second segments. If your player requires segment length matching, you can enable the `calculateSegmentLength` flag, which causes MediaTailor to automatically calculate the correct segment length by analyzing the first 10 segments from the origin and selecting the most common duration as the segment length. This flag is disabled by default and must be enabled by AWS Support. Contact AWS Support to request activation of `calculateSegmentLength` for your account or specific configurations.
- **Standard WebVTT subtitles** – For HLS workflows, MediaTailor preserves subtitle group references in the personalized multivariant playlist when WebVTT subtitles are present in the origin manifest. During ad segments, MediaTailor inserts blank WebVTT segments to maintain subtitle track continuity, because ad creatives typically do not include subtitle tracks. For DASH workflows, MediaTailor requires that subtitle adaptation sets in the origin manifest be segmented (like video and audio). Ad periods constructed by MediaTailor only contain the adaptation sets provided by the ad creative; because ad creatives typically do not include subtitle tracks, the ad periods will not contain a subtitle `AdaptationSet`. Content periods retain their subtitle adaptation sets as normal.
- **CMAF format support** – MediaTailor detects whether an HLS origin stream uses CMAF (fMP4) segments or standard TS segments and transcodes ads to match accordingly. For DASH, CMAF packaging is supported transparently, as DASH inherently uses ISO-BMFF (fMP4) as its container format. No additional configuration is required for either protocol.

 Note

MediaTailor does not support HLS streams using HVC1 (HEVC) as the codec with TS segments. Per HLS Authoring Specification for Apple Devices, section 1.5, the container format for HEVC video must be fMP4. HLS streams using HEVC must therefore use CMAF with fragmented MP4 segments.

- **Trickplay personalization** – MediaTailor automatically transcodes advertisements with trickplay variants and associated image streams (including I-frame manifests, thumbnails, and image

streams) for both HLS and DASH, matching origin content specifications. A custom transcode profile is no longer required for this capability.

- **Compact DASH manifests** – MediaTailor automatically supports compact DASH manifests, which elevate the SegmentTemplate element from individual Representation elements to the AdaptationSet level, reducing overall manifest size and improving delivery efficiency. A custom transcode profile is no longer required for this capability.

When dynamic transcoding might not be sufficient

Dynamic transcoding handles the majority of ad transcoding scenarios. However, there are some cases where additional configuration or a custom transcode profile might be needed.

- **Bitrate matching** – Dynamic transcoding does not guarantee precise bitrate matching by default. If your workflow requires ad bitrates to closely align with origin bitrates, you can enable the `enableBitrateRounding` flag, which allows for up to 5% bitrate rounding to better match origin values. This flag must be enabled by AWS Support. Contact AWS Support to request activation of `enableBitrateRounding` for your account or specific configurations. If your workflow requires bitrate matching beyond 5%, a custom transcode profile is required.
- A custom audio normalization target other than the MediaTailor default of -24 LKFS
- Non-integer segment lengths (for example, 1.92s or 3.84s)
- Dolby Vision audio
- Specific PMT, video, or audio PID values required by downstream systems or players
- Custom audio sample rates (for example, 44.1 kHz)
- Custom pixel aspect ratios (PAR) – for example, 1:1 for square pixels or 40:33 for D1/DV NTSC widescreen
- Specific named audio tracks or subtitle tracks

Custom transcode profiles

A custom transcode profile (CTP) is an explicitly authored AWS Elemental MediaConvert job configuration that you register with MediaTailor. When a CTP is associated with a MediaTailor configuration, MediaTailor uses your defined settings to transcode every ad, bumper, and slate, giving you precise control over codecs, bitrates, audio normalization levels, segment lengths, PID values, sample rates, and more. Because a CTP completely overrides dynamic transcoding, you

must explicitly configure all features you need, including trickplay features (I-frame manifests, thumbnails, and image streams), within the profile.

A CTP is designed to match a specific origin configuration. If the source content differs from the configuration the CTP was originally created for, ads might fail to transcode properly because of a `NO_VARIANT_MATCH` error. This occurs when the CTP outputs do not align with the variants present in the origin manifest. This applies to both live and VOD workflows. For example, VOD libraries that contain content from different sources or encoded with different encoding profiles (such as varying resolutions, bitrates, codecs, or segment lengths) might produce `NO_VARIANT_MATCH` errors if the CTP does not account for all variant combinations present across the library. If you need a single CTP to support multiple origins or a VOD library with mixed encoding profiles, a superset CTP can be created that encompasses the variants of all target origins. However, superset CTPs are subject to a hard limit of 36 video variants, which constrains how many origins or variant combinations can be accommodated with a single profile.

Important

You must keep your CTP in sync any time you make changes to your encoder settings or take on new sources of VOD content. Failure to do so will result in `NO_VARIANT_MATCH` errors and ad transcoding failures. This includes changes such as adding or removing resolutions, adjusting bitrates, changing codecs, modifying segment lengths, or onboarding VOD content from new providers with different encoding profiles.

When updating your CTP, be aware of the following behavior.

- Updating an existing CTP applies the new settings only to newly transcoded ads. Ads that were previously transcoded under the old CTP settings will not be retranscoded.
- Creating a new CTP and associating it with your MediaTailor configuration will prompt MediaTailor to retranscode all ads using the new profile, ensuring full consistency across your ad inventory.

For this reason, when your encoder settings change or you onboard new content sources, as a best practice, create a new CTP rather than updating the existing one to ensure all ads, including previously transcoded ads, reflect the updated settings.

When to use a custom transcode profile

Use a custom transcode profile when your workflows require one or more of the following.

- A custom audio normalization target other than the MediaTailor default of -24 LKFS
- Non-integer segment lengths (for example, 1.92 or 3.84)
- Dolby Vision audio configurations
- Specific PMT, video, or audio PID values required by downstream systems or players
- Custom audio sample rates (for example, 44.1 kHz)
- Bitrate matching beyond the 5% threshold supported by dynamic transcoding with a configuration flag
- Custom pixel aspect ratios (PAR) – for example, 1:1 for square pixels or 40:33 for D1/DV NTSC widescreen
- Specific named audio or subtitle tracks

Getting started with custom transcode profiles

Access to the MediaTailor Transcoding API, which is required to register a CTP, is restricted to allowlisted accounts. To get started, contact AWS Support using one of the following options.

- **Request allowlist access** – To create and manage your own CTPs, contact AWS Support to request that your account be added to the allowlist. After your account is added to the allowlist, you have access to the Transcoding API to register and manage profiles in any AWS region where you have an existing MediaTailor configuration.
- **Request CTP creation by AWS Support** – To have AWS Support generate and register a CTP on your behalf, contact AWS Support with a non-DRM-protected origin endpoint URL that MediaTailor can access. AWS Support will use this endpoint to analyze your origin configuration and generate an appropriate CTP for your account.

Dynamic transcoding vs. custom transcode profiles

Use the following table to determine the right approach for your workflow.

Dynamic transcoding vs. custom transcode profile comparison

Consideration	Dynamic transcoding	Custom transcode profile
Setup effort	None	Manually authored AWS Elemental MediaConvert job configuration
Trickplay support (HLS and DASH)	Automatic	Must be explicitly configured in CTP
Compact DASH manifests	Automatic	Must be explicitly configured in CTP
CMAF format support	Automatic	Must be explicitly configured in CTP
Origin change handling	Automatic	Manual CTP update required
Multi-region complexity	None	Must be registered per region
Audio normalization	-24 LKFS (default)	Fully configurable
Bitrate matching	Up to 5% rounding via <code>enableBitrateRounding</code> flag (contact AWS Support to enable)	Explicitly configured in CTP to match origin bitrates
Non-integer segment lengths	2-second default; non-integer segment lengths not supported (contact AWS Support to enable <code>calculateSegmentLength</code> for automatic integer segment length calculation)	You must manually configure segment length to match origin
Custom PID values	Not supported	Supported
Dolby Vision audio	Not supported	Supported

Consideration	Dynamic transcoding	Custom transcode profile
Custom audio sample rates	Not supported	Supported
Multi-origin support	Automatic	Superset CTP required; limited to 36 video variants
MediaTailor improvements	Automatic	Manual CTP updates required
Best suited for	Standard codec matching, common bitrate profiles, trickplay, compact DASH	Specialized encoding requirements not met by dynamic transcoding

Prefetching ads

Use AWS Elemental MediaTailor ad prefetching for live streams to help reduce peak load on ad decision servers (ADS) and decrease manifest delivery latency at the start of each ad break. When you define a prefetch schedule, MediaTailor follows the schedule to retrieve ads from the ADS and prepare them for ad insertion before they're needed for an ad break. During live streams, prefetching can help mitigate decreased ad fill rates and missed monetization opportunities because of ad request and transcoding timeouts or other network delays.

Note

Ad prefetching does not work with server-guided ad insertion (SGAI) methods, including traditional server-guided and HLS Interstitials. SGAI methods do not require prefetching because players only fetch ads they are going to play, and manifests can be served by CDNs without MediaTailor seeing individual session requests.

To set up ad prefetching, you create one or more *prefetch schedules* on your playback configuration. A prefetch schedule tells MediaTailor how and when to retrieve and prepare ads for an upcoming ad break.

- If an event has ad avails that are on a predictable schedule, use a *single prefetch schedule*. Each single prefetch schedule defines a single set of ads for MediaTailor to place in a single ad avail. To prefetch ads for multiple ad avails when you use single prefetch schedules, you must create multiple prefetch schedules (up to 24 hours before the ad avail) that correlate to each ad avail.

- If an event has ad avails that aren't on a predictable schedule, use a *recurring prefetch schedule*. A recurring prefetch schedule automatically creates a schedule and prefetches ads before each ad break in an event. The recurring prefetch schedule retrieves ads for every ad avail within a defined period of time (up to 24 hours before the event ends). You don't need to create a schedule for each ad avail, but you do lose some of the timing control that single prefetch offers.

The following topics describe more about ad prefetching.

Topics

- [How prefetching works](#)
- [Creating prefetch schedules](#)
- [TPS-based traffic shaping](#)
- [Deleting prefetch schedules](#)

How prefetching works

When your client makes a manifest request to MediaTailor, the service evaluates all of the prefetch schedules that are associated with the playback configuration. If MediaTailor doesn't find a matching prefetch schedule, the service reverts to normal ad insertion and doesn't prefetch ads.

If MediaTailor finds a matching prefetch schedule, the service evaluates the schedule based on two components: retrieval and consumption. The configuration of each component varies between single prefetch schedules and recurring prefetch schedules, as described in the following sections.

Single prefetch schedule flow

Retrieval

This defines the *retrieval window*, which is the time range when MediaTailor prefetches ads from the ADS. Be sure to schedule this window for a time that's before the ad break. The following provides an overview of how MediaTailor processes a single prefetch schedule.

For steps to create a single prefetch schedule in the console, see [Creating prefetch schedules](#). For API instruction, see [PrefetchSchedules](#) in the *AWS Elemental MediaTailor API Reference*.

During the specified *retrieval window*, MediaTailor sends requests to the ADS to retrieve and prepare ads for later insertion in playback sessions.

- MediaTailor optionally uses traffic shaping to limit the number of requests to the ADS at one time. You can choose between two approaches:

Time-window traffic shaping - MediaTailor spreads requests across the specified number of seconds instead of sending requests for all sessions at one time. This dispersed traffic distribution helps to prevent the ADS from becoming overwhelmed, resulting in time outs and low ad fill rates.

TPS-based traffic shaping - MediaTailor limits requests based on transactions per second (TPS) and concurrent users. This approach provides more intuitive configuration based on your ADS capacity limits. For more information, see [TPS-based traffic shaping](#).

- If you set up *dynamic variables*, MediaTailor includes these variables in the requests to the ADS. MediaTailor uses these variables to match ad avails to prefetch schedules during the consumption window. See the following *Consumption* section for more information.

Example

A live event lasts from 7:45AM to 10AM, with an ad break at 8:15AM. You configure MediaTailor to retrieve ads from 7:45AM to 8AM, with a traffic shaping window of 60 seconds. With 500,000 concurrent users, MediaTailor distributes the ADS requests to achieve an average rate of approximately 8,333 transactions per second for 60 seconds ($500,000 \text{ users} / 60 \text{ seconds} = 8,333$ requests per second), instead of sending all requests simultaneously.

The retrieval configuration includes the dynamic variable key `scte.event` and value 1234. MediaTailor includes this variable in the requests to the ADS, which then can be used to target specific advertisers to the event ID 1234.

Consumption

When MediaTailor encounters SCTE-35 ad break markers during the consumption window, it places the prefetched ads in an ad break.

- If you didn't set avail matching criteria, MediaTailor inserts ads in the first break in the consumption window.
- If you did set a *dynamic variable key* for *avail matching criteria*, MediaTailor evaluates these criteria against the dynamic variables that you set in the retrieval window. An ad break is eligible for prefetched ad insertion only if the avail matching criteria are met. MediaTailor inserts ads in the first break that meets the criteria.

For a list of supported avail-matching criteria, see the *Available for ad prefetch* column in the table on [MediaTailor session variables for ADS requests](#).

Example continued

You set the start time for consumption as 8:15AM, and the end time as 8:17AM. You include `scte.event_id` for the key in the avail matching criteria.

For each ad break that MediaTailor encounters from 8:15AM to 8:17AM, it evaluates the SCTE event ID for each ad break. In each playback session, MediaTailor inserts the prefetched ads in the first ad break that has an event ID of 1234 (as defined in the retrieval dynamic variables). For ad breaks that don't contain the correct event ID, MediaTailor performs standard ad insertion.

Recurring prefetch schedule flow

Retrieval

This defines the *recurring retrieval window*, which is the time range when MediaTailor prefetches and inserts ads for a live event (up to 24 hours). The following provides an overview of how MediaTailor processes recurring prefetch schedules.

For steps to create a recurring prefetch schedule in the console, see [Creating prefetch schedules](#). For API instruction, see [PrefetchSchedules](#) in the *AWS Elemental MediaTailor API Reference*.

During the specified recurring prefetch window, MediaTailor retrieves and inserts ads for a live event that's up to 24 hours. After each ad break in the window, MediaTailor automatically retrieves ads for the next ad break.

- If you set the *delay after avail end*, MediaTailor waits the specified time before retrieving the next set of ads for the next ad break.
- MediaTailor optionally uses traffic shaping to limit the number of requests to the ADS at one time. You can choose between two approaches:

Time-window traffic shaping - MediaTailor spreads requests across the specified number of seconds instead of sending requests for all sessions at one time. This dispersed traffic distribution helps to prevent the ADS from becoming overwhelmed, resulting in time outs and low ad fill rates.

TPS-based traffic shaping - MediaTailor limits requests based on transactions per second (TPS) and concurrent users. This approach provides more intuitive configuration based on your ADS capacity limits. For more information, see [TPS-based traffic shaping](#).

- If you set up *dynamic variables*, MediaTailor includes these variables in the requests to the ADS. MediaTailor uses these variables to match ad avails to prefetch schedules during the consumption window. See the following *Consumption* section for more information.

Example

A live event lasts from 7PM to 8:45PM, with four ad breaks over that time. The ad breaks aren't on a predictable schedule. You configure the recurring prefetch from 7PM to 8:45PM, with a 10 minute delay and traffic shaping window of 60 seconds. After each avail, MediaTailor retrieves ads for the next ad break. Ten minutes after the avail ends, MediaTailor starts to send retrieval requests to the ADS. With a 60-second traffic-shaping window and 500,000 concurrent users, MediaTailor distributes the ADS requests to achieve an average rate of approximately 8,333 transactions per second for 60 seconds ($500,000 \text{ users} / 60 \text{ seconds} = 8,333 \text{ requests per second}$), instead of sending all requests simultaneously.

The retrieval configuration includes the dynamic variable key `scte.event` and value 1234. MediaTailor includes this variable in the requests to the ADS, which then can be used to target specific advertisers to the event ID 1234.

Consumption

When MediaTailor encounters SCTE-35 ad break markers, it places the prefetched ads in an ad break.

- If you set the *retrieved ad expiration*, prefetched ads are available for insertion until the specified expiration.
- If you didn't set avail matching criteria, MediaTailor inserts ads in the first break in the consumption window.
- If you did set a *dynamic variable key* for *avail matching criteria*, MediaTailor evaluates these criteria against the dynamic variables that you set in the retrieval window. An ad break is eligible for prefetched ad insertion only if the avail matching criteria are met. MediaTailor inserts ads in the first break that meets the criteria.

For a list of supported avail-matching criteria, see the *Available for ad prefetch* column in the table on [MediaTailor session variables for ADS requests](#).

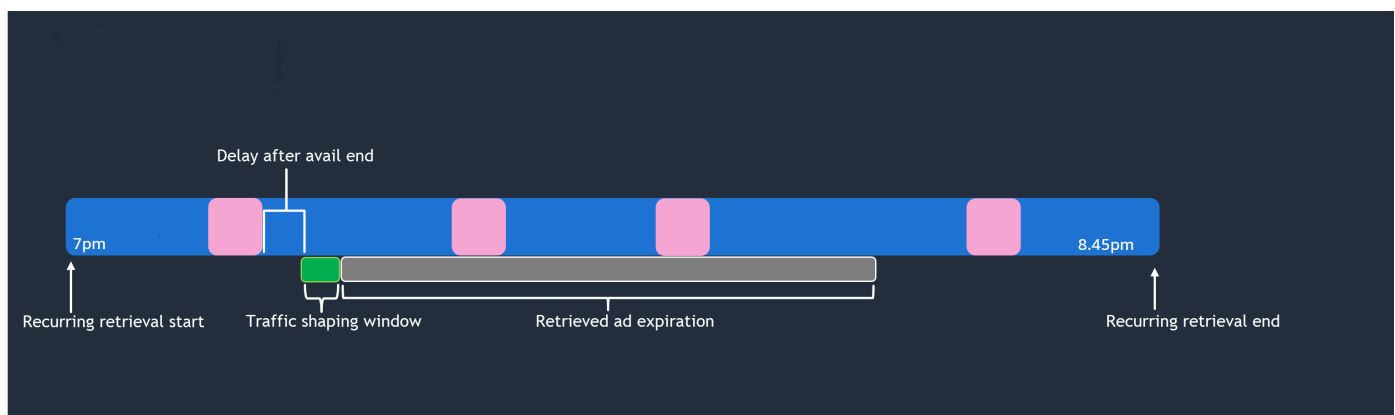
Example continued

In the consumption, you include `scte.event_id` for the key in the avail matching criteria.

For each ad break that MediaTailor encounters, it evaluates the SCTE event ID for each ad break. In each playback session, MediaTailor inserts the prefetched ads in each ad break that has an event ID of 1234 (as defined in the retrieval dynamic variables). For ad breaks that don't contain the correct event ID, MediaTailor performs standard ad insertion.

You set the ad expiration to 2700 seconds so retrieved ads are available for insertion for 45 minutes.

The following graphic illustrates the example, with the small squares representing ad breaks. The recurring prefetch schedule settings are illustrated along the event timeline.



Understanding prefetching costs

There are no costs for making ad retrieval requests. However, for prefetch ad retrieval, you'll be charged at the standard transcoding rate for the prefetched ads that MediaTailor transcodes. For prefetch ad consumption, you'll be charged at the standard rate for ad insertion for the prefetched ads that MediaTailor places in ad breaks. For information about transcoding and ad insertion costs, see [AWS Elemental MediaTailor Pricing](#).

Creating prefetch schedules

The following procedure explains how to create a prefetch schedule by using the MediaTailor console. For information about creating and managing prefetch schedules programmatically using the MediaTailor API, see [PrefetchSchedules](#) in the *AWS Elemental MediaTailor API Reference*.

 Note

When configuring prefetch schedules in MediaTailor, it's important to understand how different types of variables are handled.

Avail matching criteria

If you want to use avail matching criteria in a schedule, make sure that you first configure your playback configuration's ADS URL template with [dynamic session variables](#), otherwise the avail matching criteria won't have an effect. For information about working with dynamic session variables, see [Step 3: Configure ADS request URL and query parameters](#) in the Getting started with MediaTailor ad insertion topic.

Player variables in prefetch schedules

When you create a prefetch schedule, don't define player variables as dynamic variables in your prefetch configuration. Instead, pass the player variables as you normally would at session start. MediaTailor automatically includes these variables in prefetch ad requests if the variables are mapped in the ADS template URL.

To create a new prefetch schedule using the console

1. Open the MediaTailor console at <https://console.aws.amazon.com/mediatailor/>.
2. In the navigation pane, choose **Configurations**. Select the playback configuration that you want to create a prefetch schedule for.
3. On the **Prefetch schedules** tab, choose **Add prefetch schedule**.
4. Under the **Prefetch schedule details** pane, do the following:
 - For **Name**, enter an identifier for your prefetch schedule, such as **my-prefetch-schedule**.
 - For **Stream ID**, optionally enter a unique ID. If your origin contains multiple playback streams, you can use this ID to instruct MediaTailor to place ads in a specific stream. For example, if your playback configuration has a sports stream and a TV show stream, you can use the stream ID to create prefetch schedules to insert ads targeted for the sports stream. You pass the stream ID value to MediaTailor in your client's session initialization or manifest request. For more information see the following example.
 - For *server-side tracking*, include the `?aws.streamId` query parameter and value in your client's GET HTTP request to your MediaTailor endpoint. For general information about

server-side tracking see [MediaTailor server-side ad tracking and reporting](#). A manifest request to an HLS endpoint that includes a stream ID looks like the following, where *myStreamId* is the name of your stream ID:

```
GET <mediatailorURL>/v1/master/<hashed-account-id>/<origin-id>/<asset-id>?
aws.streamId=myStreamId
```

- For *client-side tracking*, include the `streamId` key and value in your client's POST HTTP session initialization request body to the **MediaTailor/v1/session** endpoint. For general information about client-side tracking see [Client-side ad tracking](#). A session initialization request that includes a stream ID looks like the following, where *myStreamId* is the name of your stream ID:

```
POST <mediatailorURL>/v1/session/<hashed-account-id>/<origin-id>/<asset-id>
{
  'streamId': 'myStreamId',
  'reportingMode': 'client'
}
```

5. For **Prefetch type**, make your selection and choose the corresponding section for assistance with additional fields:
 - Choose **Single** if you're creating one prefetch schedule for one ad break in an event.
 - Choose **Recurring** if you're creating a schedule that automatically prefetches ads before each ad break in an event.

Single prefetch schedule

To create a schedule that prefetches ads before one ad avail in an event.

1. On the **Retrieval** pane, specify the retrieval settings you want to use. These settings determine when MediaTailor prefetches ads from the ADS. They also determine which dynamic session variables to include in the request to the ADS, if any.
 - For **Start time**, enter the time when MediaTailor can start prefetch retrievals for this ad break. MediaTailor will attempt to prefetch ads for manifest requests made by your client on or after this time. The default value is the current time. If you don't specify a value, the service starts prefetch retrieval as soon as possible.

- For **End time**, enter the time when you want MediaTailor to stop prefetching ads for this ad break. MediaTailor will attempt to prefetch ads for manifest requests that occur at or before this time. The retrieval window can overlap with the consumption window.
- Optionally, configure traffic shaping to limit the number of requests to the ADS at one time. Choose one of the following approaches:

Time-window approach: For **Traffic shaping window duration**, enter the number of seconds that MediaTailor should distribute requests to the ADS. For more information, see [Single prefetch schedule retrieval explanation](#).

TPS-based approach: Configure **Peak TPS** and **Peak concurrent users** to limit requests based on transactions per second and concurrent users. For more information, see [TPS-based traffic shaping](#).

- In the [Dynamic variables](#) section, enter as many as 100 dynamic session variables. MediaTailor uses these variables for substitution in prefetch requests that it sends to the ADS. If you don't enter any dynamic session variables, MediaTailor makes a best effort attempt to interpolate the values for the dynamic variables contained in your [ADS](#) URL.
 - Select **Add dynamic variable**.
 - For **Key**, enter a dynamic session variable key, such as `scte.event_id`. You can use any dynamic variable that MediaTailor supports. For information about dynamic session variables, see [MediaTailor session variables for ADS requests](#).
 - For **Value**, enter a dynamic variable value, such as *my-event*.
 - To add another dynamic variable, choose Select **Add dynamic variable**.
2. On the **Consumption** pane, specify the settings that you want to use for the consumption window. These settings determine when MediaTailor places the ads into the ad break. They also determine any avail matching criteria that you want to use.
- For **Start time**, enter the time when you want MediaTailor to begin to place prefetched ads into the ad break. the default value is the current time. If you don't specify a time, the service starts prefetch consumption as soon as possible.
 - For **End time**, enter a time when you want MediaTailor to stop placing the prefetched ads into the ad break. MediaTailor will attempt to prefetch ads for your client's manifest requests that occur at or before this time. The end time must be after the start time, and less than one day from now. The consumption window can overlap with the retrieval window.

- In the [Avail matching criteria](#) section, select **Add avail criteria** and add up to five avail matching criteria to your schedule. Then, under **Dynamic variable key**, add a dynamic variable key, such as `scte.event_id`. MediaTailor will place the prefetched ads into the ad break *only* if it meets the criteria defined by the dynamic variable values that either your client passes to MediaTailor, or that MediaTailor infers from information such as session data. If an ad break doesn't meet the specified matching criteria, MediaTailor skips the prefetch for that break. For information, see the [Single prefetch schedule consumption explanation](#).

Important

You can specify a maximum of five avail matching criteria per prefetch schedule. If you specify multiple criteria, MediaTailor combines them using a logical AND. To model logical OR combinations, create multiple prefetch schedules. Only use variables that change per ad break (such as `scte.event_id`, `scte.segmentation_upid`, or `avail.index`) as avail matching criteria. Don't use player variables (such as `player_params.*`) for avail matching criteria because player variables remain constant for the entire session and don't differentiate between ad breaks. MediaTailor automatically includes player variables in prefetch ad requests if the variables are mapped in the ADS template URL.

3. Select **Add avail criteria**.

Prefetch schedules automatically expire after the consumption window's end time. For diagnostic purposes, they remain visible for at least 7 days, after which MediaTailor automatically deletes them. Alternatively, you can manually delete a prefetch schedule at any time. For information about how to manually delete a prefetch schedule, see the following [the section called "Deleting prefetch schedules"](#) section.

Determining how often your client should call the `CreatePrefetchSchedule` API

Your client can programmatically call the [CreatePrefetchSchedule](#) API once per day to set up retrieval and consumption if you have knowledge of exactly when ad breaks will occur. Or, your client can call the API many times over the course of the day to define retrieval and consumption. When choosing an API call frequency, take into consideration the [maximum number of active prefetch schedules](#), and the likelihood of whether your ad break schedule will change after you

create your prefetch schedule(s). If it's likely that the ad break schedule will change after you've created your prefetch schedule(s), you might want to call the API more frequently.

Recurring prefetch schedule

To create a schedule that prefetches ads before each ad avail in an event.

1. On the **Recurring retrieval** pane, specify the retrieval settings you want to use. These settings determine when MediaTailor prefetches ads from the ADS. They also determine which dynamic session variables to include in the request to the ADS, if any.
 - For **Recurring prefetch window**, enter the time when MediaTailor can start prefetch retrievals for this ad break. MediaTailor will attempt to prefetch ads for manifest requests made by your client on or after this time. The default value is the current time. If you don't specify a value, the service starts prefetch retrieval as soon as possible.
 - For **Delay after avail end**, enter the number of seconds that MediaTailor should wait after an avail ends before prefetching ads for the next avail. If you don't specify a value, MediaTailor defaults to a no delay.
 - Optionally, configure traffic shaping to limit the number of requests to the ADS at one time. Choose one of the following approaches:

Time-window approach: For **Traffic shaping window duration**, enter the number of seconds that MediaTailor should distribute requests to the ADS. For more information, see [Recurring prefetch schedule retrieval explanation](#)

TPS-based approach: Configure **Peak TPS** and **Peak concurrent users** to limit requests based on transactions per second and concurrent users. For more information, see [TPS-based traffic shaping](#).

- In the [Dynamic variables](#) section, enter as many as 100 dynamic session variables. MediaTailor uses these variables for substitution in prefetch requests that it sends to the ADS. If you don't enter any dynamic session variables, MediaTailor makes a best effort attempt to interpolate the values for the dynamic variables contained in your [ADS](#) URL.
 - Select **Add dynamic variable**.
 - For **Key**, enter a dynamic session variable key, such as `scte.event_id`. You can use any dynamic variable that MediaTailor supports. For information about dynamic session variables, see [MediaTailor session variables for ADS requests](#).
 - For **Value**, enter a dynamic variable value, such as *my-event*.

- To add another dynamic variable, choose Select **Add dynamic variable**.
2. On the **Consumption** pane, specify the settings that you want to use for the consumption window. These settings determine when MediaTailor places the ads into the ad break. They also determine any avail matching criteria that you want to use.
 - For **Retrieved ad expiration**, indicate how long after retrieval ads are available for insertion.
 - In the [Avail matching criteria](#) section, select **Add avail criteria** and add up to five avail matching criteria to your schedule. Then, under **Dynamic variable key**, add a dynamic variable key, such as `scte.event_id`. MediaTailor will place the prefetched ads into the ad break *only* if it meets the criteria defined by the dynamic variable values that either your client passes to MediaTailor, or that MediaTailor infers from information such as session data. If an ad break doesn't meet the specified matching criteria, MediaTailor skips the prefetch for that break. For information, see the [Recurring prefetch schedule consumption explanation](#).

 Important

You can specify a maximum of five avail matching criteria per prefetch schedule. If you specify multiple criteria, MediaTailor combines them using a logical AND. To model logical OR combinations, create multiple prefetch schedules. Only use variables that change per ad break (such as `scte.event_id`, `scte.segmentation_upid`, or `avail.index`) as avail matching criteria. Don't use player variables (such as `player_params.*`) for avail matching criteria because player variables remain constant for the entire session and don't differentiate between ad breaks. MediaTailor automatically includes player variables in prefetch ad requests if the variables are mapped in the ADS template URL.

3. Select **Add avail criteria**.

Prefetch schedules automatically expire after the consumption window's end time. For diagnostic purposes, they remain visible for at least 7 days, after which MediaTailor automatically deletes them. Alternatively, you can manually delete a prefetch schedule at any time. For information about how to manually delete a prefetch schedule, see the following [the section called "Deleting prefetch schedules"](#) section.

TPS-based traffic shaping

AWS Elemental MediaTailor provides two optional traffic shaping approaches to limit the number of requests to the ADS at one time. TPS-based traffic shaping offers an alternative to time-window based traffic shaping for prefetch schedules. This approach provides more intuitive configuration by allowing you to specify your ad decision server (ADS) capacity in terms of transactions per second (TPS) and expected concurrent users, rather than time calculations.

How TPS-based traffic shaping works

Instead of specifying retrieval window durations, you provide the following parameters:

Peak TPS

The maximum number of requests per second that your ADS can handle. This parameter has no default value.

Peak concurrent users

The expected peak number of concurrent viewers for your content. This parameter has no default value.

MediaTailor automatically distributes prefetch requests across time to stay within your specified TPS limit, regardless of the number of concurrent sessions.

Example TPS-based configuration example

Your ADS can handle 500 TPS, and you expect 100,000 concurrent viewers during peak times. You configure:

- Peak TPS: 500
- Peak concurrent users: 100,000

MediaTailor automatically distributes prefetch requests across time to stay within your specified TPS limit, regardless of the number of concurrent sessions.

Deleting prefetch schedules

The following procedure explains how to delete a prefetch schedule by using the MediaTailor console. For information about how to delete prefetch schedules programmatically using the MediaTailor API, see [DeletePrefetchSchedule](#) in the *AWS Elemental MediaTailor API Reference*.

Note

Deletion doesn't occur in real time. You might experience a delay while MediaTailor deletes the prefetch schedule(s), during which time prefetch retrieval and consumption will continue to run in the background.

To delete a prefetch schedule using the console

1. Open the MediaTailor console at <https://console.aws.amazon.com/mediatailor/>.
2. In the navigation pane, choose **Configurations**. Select the playback configuration that contains the prefetch schedule(s) that you want to delete.
3. On the **Prefetch schedules** tab, select the prefetch schedule that you want to delete. Then, choose **Delete**.

Using preconditioned ads with AWS Elemental MediaTailor

In a [typical ad insertion workflow](#), MediaTailor dynamically transcodes ads to match the content stream, saves them, and stitches the ads into the live stream. Because this process happens only after MediaTailor receives the ad in a VAST response from the ad decision server (ADS), there is a delay for when the ad is available for stitching. If additional latency is introduced to the ad stitching workflow (either because of ADS timeout or other content or network issues), MediaTailor could partially fill the avail, or miss the ad break altogether.

To reduce the time needed to stitch ads into your content, you can use preconditioned ads. A preconditioned ad is one that you transcode before using it in MediaTailor ad insertion. Instead of providing URLs for the unconditioned ads to your ADS, you provide the URLs for the preconditioned ads. In its VAST response to the MediaTailor request, the ADS includes direct links to the preconditioned ads. By removing the transcoding part of ad stitching, MediaTailor needs to just save the ad and stitch it into the content stream. The ad stitching process with

preconditioned ads reduces the time between when MediaTailor is made aware of an ad through the VAST response and when the ad is stitched into the content.

Alternatively, you can also use ad prefetching, which is when you configure MediaTailor to perform the ad stitching process at a scheduled time before the ad break is needed. For more information about ad prefetching, see [Prefetching ads](#).

Preconditioned ads requirements

The following are requirements to consider when setting up an ad stitching workflow with preconditioned ads.

MediaFiles requirements

The VAST response that the ad server sends to MediaTailor must include MediaFiles that meet these requirements:

The ad (Creative) must have variants that conform to the bitrate variants of the content stream. *It's your responsibility to ensure that the VAST response uses the right ad variants to match template manifests.*

While using preconditioned ads can help make ad insertion more efficient, MediaTailor does not have the ability to manage the transcoding process to ensure that the media files for the ads are compatible with the content manifest specifications. If the ad doesn't match the content stream, MediaTailor could miss the insertion or the mismatch could cause the playback device to error.

Additionally, to be stitched into the content stream without MediaTailor transcoding, a MediaFile must meet these requirements:

- It must be accessible on the public internet so that MediaTailor can download it.
- It must use streaming delivery, denoted as `delivery="streaming"` in the VAST response.
- It must be either a `.m3u8` (for HLS) or `.mpd` (for DASH) file.

Example VAST response

From the following example VAST response, MediaTailor inserts the MediaFile with the following URLs:

- For an HLS stream, MediaTailor uses `https://example-ad-origin.amazonaws.com/ad1/index_low.m3u8`. This is the first MediaFile with streaming delivery and a supported file extension (.m3u8).
- For a DASH stream, MediaTailor uses `https://example-ad-origin.amazonaws.com/ad1/index.mpd`. This is the first MediaFile with streaming delivery and a supported file extension (.mpd).

```
<?xml version="1.0" encoding="UTF-8"?>
<VAST xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="3.0">
  <Ad id="ad1">
    <Inline>
      <AdSystem>ExampleAdSystem</AdSystem>
      <AdTitle>ad1</AdTitle>
      <Impression><![CDATA[https://example-impression.amazonaws.com]]></
Impression>
      <AdServingId>de8e0d33-9c72-4d77-bb3a-f7e566ffc605</AdServingId>
      <Creatives>
        <Creative id="creativeId1" sequence="1">
          <Linear skipoffset="00:00:05">
            <Duration>00:00:30</Duration>
            <MediaFiles>
              <MediaFile delivery="progressive" width="1280" height="720"
type="video/mp4" bitrate="533" scalable="true" maintainAspectRatio="true"><![
CDATA[https://example-ad-origin.amazonaws.com/ad1/ad1.mp4]]></MediaFile>
              <MediaFile delivery="streaming" width="1280"
height="720" type="application/dash+xml" bitrate="533" scalable="true"
maintainAspectRatio="true"><![CDATA[https://example-ad-origin.amazonaws.com/ad1/
index.mpd]]></MediaFile>
              <MediaFile delivery="streaming" width="640"
height="360" type="application/x-mpegURL" bitrate="262" scalable="true"
maintainAspectRatio="true"><![CDATA[https://example-ad-origin.amazonaws.com/ad1/
index_low.m3u8]]></MediaFile>
              <MediaFile delivery="streaming" width="2560"
height="1440" type="application/x-mpegURL" bitrate="1066" scalable="true"
maintainAspectRatio="true"><![CDATA[https://example-ad-origin.amazonaws.com/ad1/
index_high.m3u8]]></MediaFile>
            </MediaFiles>
          </Linear>
        </Creative>
      </Creatives>
    </Inline>
```

```
</Ad>  
</VAST>
```

Ad manifest requirements

To use preconditioned ads, your parent and child ad manifests must meet these requirements:

- The manifest that's linked in the Creative section of the VAST response must be the parent ad manifest.
- The URLs for the child ad manifests must be relative paths.
- The child ad manifests must be in the same directory as the parent multivariant playlist, at the same level. Child manifests can't be in a sub-directory or other location.

Example supported parent multivariant playlist

The following parent ad multivariant playlist contains relative URLs for child ad media playlists. The child playlists are also in the same directory as the parent multivariant playlist.

```
#EXTM3U  
#EXT-X-STREAM-INF:BANDWIDTH=150000,RESOLUTION=416x234,CODECS="avc1.42e00a,mp4a.40.2"  
index_1.m3u8  
#EXT-X-STREAM-INF:BANDWIDTH=440000,RESOLUTION=416x234,CODECS="avc1.42e00a,mp4a.40.2"  
index_2.m3u8  
#EXT-X-STREAM-INF:BANDWIDTH=640000,RESOLUTION=640x360,CODECS="avc1.42e00a,mp4a.40.2"  
index_3.m3u8
```

Example unsupported parent multivariant playlist: subdirectories

The following parent ad multivariant playlist contains child playlists that are in sub-directories relative to the parent multivariant playlist. It is not a supported playlist for preconditioned ads.

```
#EXTM3U  
#EXT-X-STREAM-INF:BANDWIDTH=150000,RESOLUTION=416x234,CODECS="avc1.42e00a,mp4a.40.2"  
child/index_1.m3u8  
#EXT-X-STREAM-INF:BANDWIDTH=440000,RESOLUTION=416x234,CODECS="avc1.42e00a,mp4a.40.2"  
child/index_2.m3u8  
#EXT-X-STREAM-INF:BANDWIDTH=640000,RESOLUTION=640x360,CODECS="avc1.42e00a,mp4a.40.2"  
child/index_3.m3u8
```

Example unsupported parent multivariant playlist: absolute URLs

The following parent ad multivariant playlist contains child playlists with absolute URLs. It is not a supported playlist for preconditioned ads.

```
#EXTM3U
#EXT-X-STREAM-INF:BANDWIDTH=150000,RESOLUTION=416x234,CODECS="avc1.42e00a,mp4a.40.2"
https://example.mediatailor.us-west-2.amazonaws.com/index_1.m3u8
#EXT-X-STREAM-INF:BANDWIDTH=440000,RESOLUTION=416x234,CODECS="avc1.42e00a,mp4a.40.2"
https://example.mediatailor.us-west-2.amazonaws.com/index_2.m3u8
#EXT-X-STREAM-INF:BANDWIDTH=640000,RESOLUTION=640x360,CODECS="avc1.42e00a,mp4a.40.2"
https://example.mediatailor.us-west-2.amazonaws.com/index_3.m3u8
```

Preconditioned ads workflow

The following is a basic description of how preconditioned ads work in an ad stitching workflow with MediaTailor. The first part of the workflow are actions that you must take to get set up for using preconditioned ads. The second part describes how MediaTailor processes the ads.

Part 1: Preconditioned ads set up

Complete the following steps to set up a workflow that uses preconditioned ads in MediaTailor.

1. Use a transcoder service, such as AWS Elemental MediaConvert, to condition your creatives into variants that support the different bitrates, resolutions, and codecs of your template manifests.
2. Provide the URLs for the pre-transcoded media files to your ADS, for use in VAST responses.
3. [Create your playback](#) configuration in MediaTailor. To use preconditioned ads, select **None** for the **Streaming media file conditioning** setting in the configuration.
4. Continue with your content delivery set up as you normally would.

Part 2: MediaTailor ad processing

MediaTailor ad stitching completes as described in [How MediaTailor ad insertion works](#). When MediaTailor receives a VAST response from the ADS, it uses the following logic to determine what actions to take for the ads. This logic is dictated by the **Streaming media file conditioning** setting on the playback configuration.

- When **Streaming media file conditioning** is set to **Transcode**, MediaTailor transcodes the media files with progressive delivery and stitches them into the manifest. If there aren't enough ads

with progressive delivery media files to fill the avail, MediaTailor transcodes and uses those with streaming delivery.

- When **Streaming media file conditioning** is set to **None**, MediaTailor stitches ads with streaming delivery media files into the manifest without transcoding them. If there aren't enough ads with streaming delivery media files to fill the avail, MediaTailor transcodes and uses those with progressive delivery.

Working with MediaTailor Monetization Functions

With MediaTailor Monetization Functions (Functions), you can customize how AWS Elemental MediaTailor manages session data and builds ad requests during ad insertion. Functions let you call external APIs, transform data with expressions, and modify ad decision server (ADS) request parameters or player parameters. You don't need to deploy or manage custom infrastructure.

Functions use *JSONata*, a lightweight query and transformation language for JSON data, to evaluate expressions. You write JSONata expressions to read session data, transform values, and define output. For the complete list of supported JSONata functions, see [JSONata expression reference](#).

Use Functions when you need to:

- **Enrich ad requests with viewer data.** Call an identity service at session start and include the resolved identity in every ADS request for personalized ad targeting.
- **Customize ADS request parameters.** Dynamically set the ADS URL, headers, or body based on session data, SCTE-35 signals, or external API responses.
- **Run A/B tests across ad servers.** Split traffic between different ADS endpoints based on session attributes or random assignment.
- **Build multi-step enrichment pipelines.** Chain multiple functions together to fetch data from one API, transform it, and pass the results to the next step.

How Functions fit into the MediaTailor workflow

When a viewer starts a playback session, MediaTailor evaluates your functions at specific points in the ad insertion flow called *lifecycle hooks*.

1. You define a function and attach it to a playback configuration through a function mapping.

2. MediaTailor runs the function at the designated lifecycle hook.
3. The function reads session data, optionally calls an external API, and writes results that MediaTailor uses during playback processing.

Tip

If you are new to Functions, start with the [Quick start guide](#), then read [Lifecycle hooks](#). For expression syntax and complete examples, see [JSONata expression reference](#) and [Function examples](#).

Topics

- [Functions quick start guide](#)
- [Functions lifecycle hooks](#)
- [Function types and composition](#)
- [Creating and managing Functions](#)
- [JSONata expression reference for Functions](#)
- [Function examples](#)
- [Troubleshooting and monitoring Functions](#)
- [Functions limits](#)

Functions quick start guide

This guide walks you through creating your first function, attaching it to a playback configuration, and verifying that it ran. By the end, you will have a working function that classifies each viewer's device type (ctv, mobile, or desktop) and stores it as a player parameter available in every ADS request.

Prerequisites

Before you begin, make sure you have an existing MediaTailor playback configuration. If you don't have one, see [Getting started with MediaTailor](#).

Step 1: Create a function

In this step, you create a function that classifies the viewer's device type based on the user agent string and stores the result in player parameters. The function uses a Custom output type (no external API calls) with a [JSONata expression reference](#) expression to evaluate the user agent.

1. Open the MediaTailor console at <https://console.aws.amazon.com/mediatailor/>.
2. In the navigation pane, choose **Functions**.
3. Choose **Create function**.
4. In the create wizard modal, select **Create from scratch**, then choose **Continue**.
5. Under **Function type**, select the **Custom output** tile.
6. Under **Function details**, enter the following:
 - **Function ID:** myFirstFunction
 - **Description:** Classify device type from user agent
7. Under **Custom output configuration**, in the **Output** section, add one row:
 - **Key:** player_params.deviceType
 - **Value:** {% \$contains(session.user_agent, 'CTV') ? 'ctv' : \$contains(session.user_agent, 'Mobile') ? 'mobile' : 'desktop' %}
8. Choose **Create function**.

A success notification confirms the function was created, and you are redirected to the function details page.

The resulting function configuration is:

```
{
  "FunctionId": "myFirstFunction",
  "FunctionType": "CUSTOM_OUTPUT",
  "Description": "Classify device type from user agent",
  "CustomOutputConfiguration": {
    "Runtime": "JSONATA",
    "Output": {
      "player_params.deviceType": "{% $contains(session.user_agent, 'CTV') ?
'ctv' : $contains(session.user_agent, 'Mobile') ? 'mobile' : 'desktop' %}"
    }
  }
}
```

Step 2: Attach the function to a playback configuration

Map the function to a lifecycle hook on your playback configuration. The mapping tells MediaTailor when to run the function.

1. In the navigation pane, choose **Configurations**.
2. Choose the playback configuration you want to update.
3. Choose **Edit**.
4. Expand the **Functions configuration** section.
5. For **Session initialization hook**, select `myFirstFunction` from the dropdown.
6. Choose **Save**.

This attaches `myFirstFunction` to the [PRE_SESSION_INITIALIZATION](#) lifecycle hook. The resulting function mapping is:

```
{
  "FunctionMapping": {
    "PRE_SESSION_INITIALIZATION": "myFirstFunction"
  }
}
```

MediaTailor runs the function once at the start of every new session on this playback configuration.

Step 3: Start a session and verify the function ran

Start a new playback session to trigger the function. Use a session initialization request to your playback configuration's session initialization endpoint.

MediaTailor automatically publishes CloudWatch metrics for every function execution — no opt-in required. After starting a session, check the following metrics in the `AWS/MediaTailor` namespace to confirm your function ran:

- `PreSessionInitHook.Invocations` — Confirms the hook fired.
- `PreSessionInitHook.Errors` — Should be 0 if the function succeeded.
- `Function.Invocations` — Confirms the individual function executed. This metric includes `FunctionId`, `FunctionType`, and `HookType` dimensions so you can filter to `myFirstFunction` specifically.

If the function fails, MediaTailor emits error log events to Manifest Logs by default (no configuration needed):

- `PRE_SESSION_INIT_HOOK_ERROR` — Hook-level failure with `errorType` and `cause`.
- `PRE_SESSION_INIT_FUNCTION_ERROR` — Function-level failure with the specific `functionId` and error details.

The following example shows a `PRE_SESSION_INIT_FUNCTION_ERROR` event for a syntax error in the function expression:

```
{
  "eventTimestamp": "2024-01-01T12:00:00.076000000Z",
  "eventType": "PRE_SESSION_INIT_FUNCTION_ERROR",
  "eventDescription": "Function execution failed",
  "awsAccountId": "123456789012",
  "originId": "my-config",
  "sessionId": "session-123",
  "requestId": "req-abc",
  "eventId": "5dc6f040-0f72-4e8c-a64e-25eeef62708c",
  "functionId": "myFirstFunction",
  "functionType": "CUSTOM_OUTPUT",
  "executionTimeMs": 2,
  "errorType": "SYNTAX_ERROR",
  "cause": "Expected \"\\\"\" before end of expression",
  "input": {}
}
```

Use the `eventId` field to correlate hook and function error events for the same execution. The `errorType` field tells you the class of failure — see [Troubleshooting and monitoring](#) for a full list of error types and fixes.

Note

For detailed success logging, opt in to `PRE_SESSION_INIT_HOOK_SUMMARY` and `PRE_SESSION_INIT_FUNCTION_COMPLETED` events in your Manifest Log configuration. Summary events show the hook outcome for every execution. Completed events show each function's input, output, and HTTP request/response details. These are disabled by default to minimize log costs. For more information, see [Troubleshooting and monitoring](#).

What happens behind the scenes

Here is the complete request flow for the function you just created:

1. The player initiates a session with MediaTailor.
2. MediaTailor fires the `PRE_SESSION_INITIALIZATION` lifecycle hook and runs `myFirstFunction`.
3. The function evaluates the `session.user_agent` field and writes `ctv`, `mobile`, or `desktop` to `player_params.deviceType`.
4. MediaTailor creates the session and returns the manifest to the player.
5. The player encounters an ad break during playback.
6. MediaTailor fires the `PRE_ADS_REQUEST` lifecycle hook, then constructs the ADS request. Because `deviceType` is stored in player parameters, it is available for inclusion in the ADS request URL through dynamic variable substitution.
7. The ADS uses the device type to return targeted ad creatives.
8. MediaTailor stitches the ads into the manifest and returns it to the player.

If the function fails for any reason, MediaTailor discards the output and proceeds as if no function were attached. The viewer still sees ads — just without the device type targeting.

Suggested topics

You now have a working function attached to a playback configuration. From here:

- To learn what input fields and output namespaces are available at each lifecycle hook, see [Lifecycle hooks](#).
- To learn about the different function types and how to chain them together, see [Function types and composition](#).
- To see complete working examples, see [Function examples](#).

Functions lifecycle hooks

A lifecycle hook defines when MediaTailor runs your function during playback. This page is a complete reference for input fields, output namespaces, and the rules that govern data flow at each hook.

Overview

MediaTailor supports two lifecycle hooks:

- **PRE_SESSION_INITIALIZATION** fires once when a viewer starts a new session. Use it for one-time setup work such as fetching audience segments. At this point, no ad break has occurred, so ad break context is not available.
- **PRE_ADS_REQUEST** fires before every ad decision server (ADS) request — once per ad break in the stream. Use it to customize the ADS request with targeting data, modify the ADS URL, or add headers.

The key difference is timing: **PRE_SESSION_INITIALIZATION** runs once and sets up data that persists for the entire session, while **PRE_ADS_REQUEST** runs repeatedly and can tailor each ADS request to the specific ad break.

PRE_SESSION_INITIALIZATION

When it fires

MediaTailor runs the function mapped to **PRE_SESSION_INITIALIZATION** once, at the start of a new playback session. The function runs before MediaTailor constructs the initial manifest response.

Input

`session.*`, `player_params.*`, and `event.*`. For all available fields, see [Input field reference](#).

Output namespace allowed

Namespace	Accepted types
<code>player_params.*</code>	Strings, numbers, booleans

Values written to `player_params.*` are persisted to the session. They are available:

- As input at the **PRE_ADS_REQUEST** lifecycle hook via `player_params.*`
- In ADS request URLs through [MediaTailor dynamic ad variables for ADS requests](#) (for example, `[player_params.deviceType]`)

- For the lifetime of the session across all ad breaks

Note

The total serialized size of all `player_params` output keys and values must not exceed 1,000 characters. If the total exceeds this limit, the function output is discarded. For more information, see [Functions limits](#).

Typical use cases

- Fetch identity or audience data from an external service and store it in player parameters for use in later ADS requests.
- Classify the device type based on the user agent and write the classification to a player parameter.
- Set default player parameter values that downstream ad break processing relies on.
- Store values in player parameters that are included in the ADS URL through [MediaTailor dynamic ad variables for ADS requests](#).

Failure behavior

If a function attached to `PRE_SESSION_INITIALIZATION` fails for any reason, MediaTailor discards the function's output and proceeds as if no function were attached. The session starts normally without the function's player parameter values.

PRE_ADS_REQUEST

When it fires

MediaTailor runs the function mapped to `PRE_ADS_REQUEST` once per ad break, immediately before sending the request to the ADS. The function runs each time an ad opportunity is encountered during manifest processing.

Input

All fields from `PRE_SESSION_INITIALIZATION`, plus `avail.*`, `scte.*`, `asset.*`, and `adsRequest.*` (url, method, headers, body). For all available fields, see [Input field reference](#).

Output namespace allowed

Namespace	Accepted types	How the output is used
<code>player_params.*</code>	Strings, numbers, booleans	Overrides session player parameters for this ad break. Available to the ADS request URL through MediaTailor dynamic ad variables for ADS requests .
<code>session.*</code>	Strings, numbers, booleans	Overrides session variables for this ad break. Available to the ADS request URL through dynamic variable substitution.
<code>avail.*</code>	Strings, numbers, booleans	Overrides avail variables for this ad break. Available to the ADS request URL through dynamic variable substitution.
<code>scte.*</code>	Strings, numbers, booleans	Overrides SCTE variables for this ad break. Available to the ADS request URL through dynamic variable substitution.
<code>asset.*</code>	Strings, numbers, booleans	Overrides asset metadata variables (from EXT-X-ASSET tags) for this ad break. Available to the ADS request URL through dynamic variable substitution. For more information about HLS ad markers, see HLS ad markers .
<code>adsRequest.*</code>	String	Overrides the ADS request for this ad break only.

Namespace	Accepted types	How the output is used
		Supported fields: <code>url</code> , <code>method</code> , <code>headers.<name></code> , <code>body</code> . The <code>url</code> value is treated as a template and supports MediaTailor dynamic ad variables for ADS requests after the function runs. Not persisted.

Note

All output from the `PRE_ADS_REQUEST` hook are transient overrides — they apply only to the current ad break's ADS request and are not persisted to the session.

Example — rewriting the ADS request:

```
{
  "Output": {
    "adsRequest.url": "{%'https://ads.example.com/v1/vast?sid=' & session.id &
'&genre=' & player_params.genre%}",
    "adsRequest.headers.X-Custom-Token": "{%player_params.auth_token%}"
  }
}
```

This replaces the ADS URL and adds a custom header for the current ad break.

Typical use cases

- Rewrite the ADS request URL to route traffic between different ad servers for A/B testing.
- Append enrichment data (audience segments, identity tokens) to the ADS request URL or headers.
- Conditionally modify ADS request parameters based on SCTE-35 signal data or avail index.

Failure behavior

If a function attached to PRE_ADS_REQUEST fails for any reason, MediaTailor discards the function's output and proceeds as if no function were attached. The ADS request is sent using the original session and request parameters without modification.

Input field reference

Field	Type	PRE_SESSION_INITIALIZATION	PRE_ADS_REQUEST
session.id	Long	✓	✓
session.uuid	String	✓	✓
session.client_ip	String	✓	✓
session.user_agent	String	✓	✓
session.referer *	String	✓	✓
session.avail_duration_secs	Long	✗	✓
session.avail_duration_ms	Long	✗	✓
player_params.*	String	✓	✓
event.id	String	✓	✓
event.hook	String	✓	✓
event.timestamp	String	✓	✓
avail.index	Int	✗	✓

Field	Type	PRE_SESSION_INITIALIZATION	PRE_ADS_REQUEST
avail.random	Long	X	✓
avail.source_content_time_epoch_ms	Long	X	✓
scte.event_id	Int	X	✓
scte.avail_num	Int	X	✓
scte.segmentation_event_id	Int	X	✓
scte.segmentation_type_id	Int	X	✓
scte.segmentation_upid	String	X	✓
scte.segmentation_upid.assetId	String	X	✓
scte.segmentation_upid.cueData.key	String	X	✓
scte.segmentation_upid.cueData.value	String	X	✓

Field	Type	PRE_SESSION_INITIALIZATION	PRE_ADS_REQUEST
scte.unique_program_id	Int	X	✓
scte.archive_allowed_flag	Boolean	X	✓
scte.deliver_not_restricted_flag	Boolean	X	✓
scte.device_restrictions	Int	X	✓
scte.no_regional_blackout_flag	Boolean	X	✓
scte.segment_num	Int	X	✓
scte.segments_expected	Int	X	✓
scte.sub_segment_num	Int	X	✓
scte.sub_segments_expected	Int	X	✓
scte.avails_expected	Long	X	✓

Field	Type	PRE_SESSION_INITIALIZATION	PRE_ADS_REQUEST
<code>asset.*</code>	String	X	✓
<code>adsRequest.url</code>	String	X	✓
<code>adsRequest.method</code>	String	X	✓
<code>adsRequest.headers.<key></code>	String	X	✓
<code>adsRequest.body</code>	String	X	✓

* `session.referrer` is only present when a Referer header is included in the session initialization request. Use `$exists(session.referrer)` to check before accessing.

Function types and composition

AWS Elemental MediaTailor provides three function types, each designed for a different category of work. You choose a type when you create a function, and the type determines what the function can do at runtime. This page explains how each type executes, when to use it, and how to compose functions into multi-step pipelines.

Function type overview

Type	Category	Purpose
CUSTOM_OUTPUT	Data transformation	Evaluate expressions against the current session state and produce outputs. No external calls.
HTTP_REQUEST	External integration	Make an HTTP call to an external service, then evaluate output expressio

Type	Category	Purpose
		ns that can reference the response.
SEQUENTIAL_EXECUTOR	Orchestration	Run a sequence of functions in order, passing data between steps through temporary data.

Each type serves a distinct role. Choose a type when you create a function — the type determines what the function can do at runtime.

CUSTOM_OUTPUT

When to use

Use CUSTOM_OUTPUT when your function logic is purely computational. Common use cases include classifying devices based on user agent strings, selecting ad decision server URLs with A/B testing logic, and reformatting player parameters before they reach the ad decision server.

CUSTOM_OUTPUT functions are the fastest type because they evaluate expressions in memory with no network calls. Choose this type whenever you can accomplish your goal without contacting an external service.

Configuration fields

A CUSTOM_OUTPUT function has two fields:

- **Runtime** — The expression language. Set this to JSONATA.
- **Output** — A map of key-value pairs. Each key is an output binding name (such as `player_params.device_type`), and each value is an expression that MediaTailor evaluates at runtime.

Note

Use `{%...%}` to wrap JSONata expressions in string fields. Values without these delimiters are treated as static strings. For example, `"{%session.id%}"` evaluates the expression, while `"session.id"` is the literal text.

Runtime behavior

When MediaTailor executes a `CUSTOM_OUTPUT` function, it evaluates every expression in the output block against the current session state. All expressions see the same input snapshot. The evaluated results become the function's outputs.

If any expression fails to evaluate, MediaTailor discards the entire function's output and proceeds as if no function were attached. The function does not partially commit results.

Example: A/B traffic split

The following function randomly assigns each ad break to one of two ad decision server URLs. It is designed for the `PRE_ADS_REQUEST` lifecycle hook.

```
{
  "FunctionId": "abTestAdsUrl",
  "FunctionType": "CUSTOM_OUTPUT",
  "CustomOutputConfiguration": {
    "Runtime": "JSONATA",
    "Output": {
      "adsRequest.url": "{%$random() < 0.5 ? 'https://ads.example.com/v1' : 'https://ads.example.com/v2'%}"
    }
  }
}
```

For a complete walkthrough of this example, see [Function examples](#).

HTTP_REQUEST**When to use**

Use `HTTP_REQUEST` when your function needs to call an external service. Common use cases include fetching identity data from a resolution provider, retrieving audience segments from a data management platform, and sending session information to a logging endpoint.

Configuration fields

An HTTP_REQUEST function has the following fields:

- **Runtime** — The expression language. Set this to JSONATA.
- **MethodType** — The HTTP method. Supported values are GET and POST.
- **Url** — The URL to send the request to. You can use a static URL or a JSONata expression that builds the URL dynamically.
- **Headers** — The HTTP headers to include in the request, specified as header name and value pairs. Use `{% . . %}` expression syntax for dynamic header values. Static values can be specified directly as strings.
- **Body** — The request body to send. Used with POST requests. You can use a JSONata expression to build the body dynamically.
- **RequestTimeoutMilliseconds** (required) — How long to wait for a response.
- **Output** — Defines the values to produce after the HTTP call completes. Each entry maps an output key (such as `player_params.envelope_id`) to an expression that can reference the response object.

For size limits and restrictions that apply to these fields, see [Limits](#).

How the request is processed

MediaTailor processes an HTTP_REQUEST function in two steps:

1. **Build the request** — MediaTailor evaluates the `Url`, `Headers`, and `Body` expressions against the current session state. These evaluated values form the outbound HTTP request.
2. **Process the response** — After the HTTP call completes, MediaTailor evaluates the expressions in the output block. These expressions can reference both the original session state and the response object returned by the call.

Response fields

After the HTTP call completes, you can reference the following fields in your Output expressions:

Field	Type	Description
<code>response.body</code>	Object or Array	The response body parsed as JSON. Set to <code>null</code> if the body exceeds 20,000 characters or is not valid JSON.
<code>response.statusCode</code>	Integer	The HTTP status code returned by the external service. Set to <code>null</code> on network failure.
<code>response.text</code>	String	The raw response body as a string, truncated to 20,000 characters. Set to "Internal Error" on network failure.

** Important**

The `response.body` field is `null` when the response exceeds 20,000 characters, even if the response is valid JSON.

** Note**

The response object is available only in the Output block of an `HTTP_REQUEST` function. You cannot reference response fields in the Url, Headers, or Body fields. In a `SEQUENTIAL_EXECUTOR`, each `HTTP_REQUEST` function can access only its own response.

A value of `null` means the data is not available. This happens when the HTTP call fails (network error or timeout) or when the response body exceeds 20,000 characters or is not valid JSON.

Network failure behavior

If the HTTP call fails due to a network error or timeout, `response.statusCode` and `response.body` are set to `null`, and `response.text` is set to "Internal Error". Your Output expressions still run, so always check `response.statusCode` before using response data.

Tip

Use a conditional expression to handle failures gracefully: `{%response.statusCode = 200 ? response.body.value : 'default'%}`

Example: Fetch identity data

The following function calls an identity resolution API at session start and stores the result in player parameters. It is designed for the `PRE_SESSION_INITIALIZATION` lifecycle hook.

```
{
  "FunctionId": "fetchIdentityEnvelope",
  "FunctionType": "HTTP_REQUEST",
  "HttpRequestConfiguration": {
    "Runtime": "JSONATA",
    "MethodType": "GET",
    "Url": "{%'https://identity.example.com/v1/resolve?ip=' &
$encodeURIComponent(session.client_ip)%}'",
    "Headers": {
      "Authorization": "{%'Bearer my_api_token'%}'",
      "Accept": "application/json"
    },
    "RequestTimeoutMilliseconds": 2000,
    "Output": {
      "player_params.identity_envelope": "{%response.statusCode = 200 ?
response.body.envelope : ''%}"
    }
  }
}
```

For a complete walkthrough of a similar example, see [Function examples](#).

SEQUENTIAL_EXECUTOR

When to use

A `SEQUENTIAL_EXECUTOR` runs a series of functions one at a time, in order. Each step can use the results of the steps before it.

Use `SEQUENTIAL_EXECUTOR` when your logic requires multiple steps that depend on each other's results. Common use cases include fetching identity data and then using it to retrieve audience segments, running a classification step and then conditionally calling different external services based on the result, and building complex ad request URLs from multiple data sources.

Configuration fields

A `SEQUENTIAL_EXECUTOR` function has the following fields:

- **Runtime** — The expression language. Set this to `JSONATA`.
- **FunctionList** — An ordered list of 1 to 10 steps. Each step specifies the `FunctionId` of the function to run. Optionally, you can add a `RunCondition` expression to control whether the step runs or is skipped.
- **Output** — Defines the values to produce after all steps complete. Each entry maps an output key (such as `player_params.envelope`) to an expression that can reference data produced by any step in the sequence. If omitted, all output from the individual functions in the sequence is used.
- **TimeoutMilliseconds** (required) — The maximum time for the entire sequence to complete. If the sequence exceeds this timeout, MediaTailor discards all output from the sequence.

Ordered execution and data flow

MediaTailor runs each step in the sequence from first to last. After each step completes, the values it produces are merged into a running set of results. Subsequent steps can access the original session data plus all values produced by earlier steps.

Temporary data is the primary mechanism for passing data between steps. When a function writes to a `temp.*` key, the next step can read that value. Player parameters and ad request fields written by earlier steps are also visible to later steps.

Note

Temporary data accepts any data type, including objects and arrays. Player parameters and ad request fields accept only strings, numbers, booleans, and null).

Per-step run conditions

Each step in the sequence has an optional `RunCondition` field. This field contains an expression that returns `true` or `false`. MediaTailor evaluates the `RunCondition` expression immediately before running that step.

If the `RunCondition` expression evaluates to `false`, MediaTailor skips the step entirely and moves to the next one. If the `RunCondition` field is omitted, the step always runs.

```
{ "FunctionId": "retryFetch", "RunCondition": "{%temp.statusCode = 500%}" }
```

This mechanism lets you build conditional pipelines. For example, you can run an identity fetch in step 1, then conditionally run a segment lookup in step 2 only if step 1 returned a valid identity.

How the output block works

The output block on a `SEQUENTIAL_EXECUTOR` controls what the sequence produces after all steps complete:

- **Output block present** — MediaTailor evaluates the expressions in the output block against the final accumulated state and saves only those outputs. Any outputs produced by earlier steps that are not referenced in the sequential output block are discarded.
- **Output block absent** — MediaTailor saves all accumulated outputs from all steps directly.

Tip

Omit the output block when you want every function's output to pass through. Add an output block when you need to filter, rename, or transform the accumulated results before saving them.

Timeout configuration

The `TimeoutMilliseconds` field sets a deadline for the entire sequence. This timeout covers all steps, including any HTTP calls made by functions. If the sequence exceeds the timeout, MediaTailor discards all output from the sequence and proceeds as if no function were attached.

Individual `HTTP_REQUEST` functions still respect their own `RequestTimeoutMilliseconds` setting. The sequence timeout acts as an outer boundary that caps the total execution time.

Example: Retry on HTTP failure

This example calls an identity API and automatically retries if the first call returns a server error. It uses two `HTTP_REQUEST` functions orchestrated by a `SEQUENTIAL_EXECUTOR`.

Step 1 — Primary fetch (`fetchIdentity`):

```
{
  "FunctionId": "fetchIdentity",
  "FunctionType": "HTTP_REQUEST",
  "HttpRequestConfiguration": {
    "Runtime": "JSONATA",
    "MethodType": "GET",
    "Url": "{%'https://identity.example.com/v1/resolve?ip=' & session.client_ip%}'",
    "Headers": {
      "Accept": "application/json"
    },
    "RequestTimeoutMilliseconds": 1000,
    "Output": {
      "temp.statusCode": "{%response.statusCode%}",
      "temp.envelope": "{%response.statusCode = 200 ? response.body.envelope :
null%}"
    }
  }
}
```

Step 2 — Retry on failure (`retryIdentity`):

```
{
  "FunctionId": "retryIdentity",
  "FunctionType": "HTTP_REQUEST",
  "HttpRequestConfiguration": {
    "Runtime": "JSONATA",
```

```

    "MethodType": "GET",
    "Url": "{%'https://identity-fallback.example.com/v1/resolve?ip=' &
session.client_ip%}",
    "Headers": {
        "Accept": "application/json"
    },
    "RequestTimeoutMilliseconds": 1000,
    "Output": {
        "temp.statusCode": "{%response.statusCode%}",
        "temp.envelope": "{%response.statusCode = 200 ? response.body.envelope :
null%}"
    }
}

```

Sequence (**identityWithRetry**):

```

{
    "FunctionId": "identityWithRetry",
    "FunctionType": "SEQUENTIAL_EXECUTOR",
    "SequentialExecutorConfiguration": {
        "Runtime": "JSONATA",
        "TimeoutMilliseconds": 2000,
        "FunctionList": [
            { "FunctionId": "fetchIdentity" },
            { "FunctionId": "retryIdentity", "RunCondition": "{%temp.statusCode >=
500%}" }
        ],
        "Output": {
            "player_params.envelope": "{%temp.envelope%}"
        }
    }
}

```

How it works:

1. `fetchIdentity` calls the identity API and writes the status code and envelope to `temp.*`.
2. If the status code is 500 or higher, the `RunCondition` on step 2 evaluates to true and `retryIdentity` runs. It overwrites `temp.statusCode` and `temp.envelope` with the retry response.
3. If the first call succeeded, step 2 is skipped.

4. The sequence Output block writes `temp.envelope` to `player_params.envelope`.

Composition rules

MediaTailor enforces the following limits on function composition:

Rule	Limit
Maximum nesting depth	2 (a sequence can contain functions, but those children cannot be sequences themselves)
Steps per sequence	1 to 10
Total function executions per lifecycle hook	20
Circular references	Not allowed. A function cannot reference itself, directly or indirectly.
Function existence	All functions referenced in a <code>FunctionList</code> must exist before you create or update the parent sequence.

When you create a function, MediaTailor validates expression syntax, checks for restricted functions, verifies that all referenced functions exist, and detects circular references. When you attach a function to a playback configuration, MediaTailor additionally validates that all output keys across the entire function tree are compatible with the assigned lifecycle hook.

Creating and managing Functions

This page walks you through creating, attaching, updating, and deleting Functions using the MediaTailor console. Functions let you customize session behavior and ad requests by running logic at key points during playback.

Setting up a function

Creating a function using the console

1. Open the MediaTailor console at <https://console.aws.amazon.com/mediatailor/>.

2. In the navigation pane, choose **Functions**.
3. Choose **Create function**.
4. For **Function ID**, enter a unique name for the function (for example, `fetchIdentity`).
5. For **Function type**, choose `CUSTOM_OUTPUT`, `HTTP_REQUEST`, or `SEQUENTIAL_EXECUTOR`. The console displays the configuration fields for the selected type.
6. Fill in the type-specific configuration fields. For a description of each function type and its fields, see [Function types and composition](#).
7. Choose **Create function**.

Attaching a function to a playback configuration

A function does not run until you attach it to a playback configuration through a function mapping (a link between a lifecycle hook and a function). The function mapping specifies which lifecycle hook triggers the function.

1. Open the MediaTailor console.
2. In the navigation pane, choose **Configurations**.
3. Choose the playback configuration you want to update.
4. In the **Function mapping** section, choose **Edit**.
5. For each lifecycle hook, select the function to attach:
 - **Session initialization hook** — Choose a function to run once at session start.
 - **Ad request hook** — Choose a function to run before each ADS request.
6. Choose **Save**.

Managing functions

Viewing functions

Navigate to **Functions** to see all functions in your account. Choose a function name to view its configuration.

Updating an existing function

To update a function, navigate to **Functions**, choose the function, and modify the configuration. The update replaces the entire function definition.

Tip

Save a copy of your function configuration before making changes. There is no built-in versioning or rollback for functions.

Removing a function

Detaching a function from a playback configuration

Before you delete a function, remove it from all playback configurations that reference it.

1. Open the MediaTailor console.
2. In the navigation pane, choose **Configurations**.
3. Choose the playback configuration.
4. In the **Function mapping** section, choose **Edit**.
5. Remove the function from the lifecycle hook.
6. Choose **Save**.

Deleting a function

Navigate to **Functions**, select the function, and choose **Delete**.

Deletion blocking rules

MediaTailor prevents you from deleting a function that is still in use.

Condition	Result
Function is attached to a playback configuration via function mapping	Delete is blocked. Detach the function first.
Function is referenced in a SEQUENTIAL_EXECUTOR	Delete is blocked. Remove the reference from the parent function first.
Function is not referenced anywhere	Delete succeeds.

Validation rules

MediaTailor validates your function when you create or update it. The following checks are performed:

- **Expression syntax** — All expressions must be valid JSONata.
- **Restricted functions** — Expressions cannot call restricted JSONata functions. See [JSONata expression reference](#) for the full list.
- **Output key prefixes** — All output keys must start with a recognized namespace prefix. For the list of accepted prefixes, see [Lifecycle hooks](#).
- **Function references** — All function IDs in a `FunctionList` must reference existing functions.
- **Circular references** — A function cannot reference itself, directly or indirectly.
- **Nesting depth** — A `SEQUENTIAL_EXECUTOR` can call other functions, but those functions cannot themselves be `SEQUENTIAL_EXECUTORs`.

For specific values and size limits, see [Limits](#).

API reference

To manage functions programmatically, see the [AWS Elemental MediaTailor API Reference](#).

JSONata expression reference for Functions

This page is a complete reference for the expression syntax, operators, and functions available in Functions. Use it when you write expressions for output blocks, URL fields, header values, body templates, and run conditions.

Expression delimiters

Every value you define in a function is either a constant or an expression — not a mix of both. MediaTailor distinguishes between the two based on delimiters.

Syntax	Type	Evaluation
<code>https://ads.example.com/vast</code>	Constant	Returned as-is with no evaluation.

Syntax	Type	Evaluation
<code>{%session.client_ip %}</code>	Expression	Evaluated at runtime. The result replaces the entire value.
GET	Constant	Returned as-is.
<code>{%'https://ads.example.com/vast?ip=' & session.client_ip%}</code>	Expression	Evaluated at runtime.

Important

A value is either entirely a constant or entirely an expression. You cannot mix the two in a single value. For example, `hello {% 'world' %}` is not valid. To combine static text with dynamic values, use string concatenation inside the expression: `{%'hello ' & 'world' %}`.

Language basics

Dot notation for path navigation

Use dot notation to traverse the input data. Each dot descends one level into the object hierarchy.

```
session.client_ip           # the viewer's IP address
response.body.envelope      # a field inside a parsed JSON response
player_params.campaign_id   # a player parameter
```

Missing fields return `null` without raising an error:

```
temp.nonExistent           # null
temp.nonExistent.deeply.nested # null
```

String concatenation with &

The `&` operator joins two string values. Non-string values are converted to strings automatically.


```
'https://ads.example.com/vast?ip=' & session.client_ip  
# "https://ads.example.com/vast?ip=192.0.2.1"  
  
'duration=' & 30  
# "duration=30"
```

Conditional (ternary) expressions

Use the ternary operator to return one of two values based on a condition.

```
condition ? value_if_true : value_if_false
```

Examples:

```
$exists(player_params.env) ? player_params.env : 'prod'  
response.statusCode = 200 ? response.body.id : 'unknown'  
$random() > 0.5 ? 'groupA' : 'groupB'
```

You can nest ternary expressions for multi-way branching:

```
$contains(session.user_agent, 'CTV') ? 'ctv'  
  : $contains(session.user_agent, 'Mobile') ? 'mobile'  
  : 'desktop'
```

Variable binding with :=

Use the := operator inside parentheses to assign intermediate values within an expression. Bound variables are scoped to the enclosing parentheses and do not persist outside the expression.

```
(  
  $base := 'https://ads.example.com';  
  $base & '/vast?ip=' & session.client_ip  
)
```

Semicolons separate statements inside parentheses. The last statement is the return value of the expression.

```
(  
  $code := response.statusCode;  
  $code != null and $code >= 200 and $code < 300  
)
```

```
? response.body.value  
: 'fallback'  
)
```

Operators

Arithmetic

Operator	Description	Example	Result
+	Addition	5 + 3	8
-	Subtraction	10 - 4	6
*	Multiplication	6 * 7	42
/	Division	15 / 4	3.75
%	Modulo	17 % 5	2

Important

Input values from player parameters and session data arrive as strings. Use `$number()` to convert them before numeric comparisons or arithmetic. Comparing a string to a number produces unexpected results.

Comparison

Operator	Description	Example	Result
=	Equal to	<code>response. statusCode = 200</code>	true
!=	Not equal to	<code>player_pa rams.region != 'us-east-1'</code>	true if not us-east-1

Operator	Description	Example	Result
<	Less than	<code>avail.index < 3</code>	true if below 3
>	Greater than	<code>\$number(player_params.age) > 18</code>	true if over 18
<=	Less than or equal	<code>\$count(items) <= 10</code>	true if 10 or fewer
>=	Greater than or equal	<code>response.statusCode >= 400</code>	true if error status

Boolean

Operator	Description	Example
and	Logical AND	<code>response.statusCode = 200 and \$exists(response.body.id)</code>
or	Logical OR	<code>player_params.region = 'us-east-1' or player_params.region = 'us-west-2'</code>

Use parentheses for precedence:

```
score > 0.5 and (tier = 'premium' or tier = 'gold')
```

Note

Use `$not()` for logical negation. There is no `not` keyword operator.

Membership (in)

The `in` operator tests whether a value exists in an array.

```
'premium' in segments      # true if segments contains 'premium'
player_params.region in ['us-east-1', 'us-west-2'] # true
```

Chaining (~>)

The chaining operator passes the result of the left-hand expression as the first argument to the function on the right.

```
session.user_agent ~> $lowercase ~> $trim
# equivalent to $trim($lowercase(session.user_agent))
```

Allowed functions

MediaTailor supports the following built-in functions. Any function not listed here is blocked and causes a validation error when you create or update a function.

Type conversion (3)

Function	Description	Example	Result
<code>\$string(value)</code>	Convert to string	<code>\$string(200)</code>	"200"
<code>\$number(value)</code>	Convert to number	<code>\$number('42')</code>	42
<code>\$boolean(value)</code>	Convert to boolean	<code>\$boolean(1)</code>	true

Introspection (4)

Function	Description	Example	Result
<code>\$length(string)</code>	String length	<code>\$length('hello')</code>	5
<code>\$count(array)</code>	Array element count	<code>\$count([1, 2, 3])</code>	3

Function	Description	Example	Result
<code>\$exists(value)</code>	Check if value exists (not undefined)	<code>\$exists(emp.id)</code>	true or false
<code>\$keys(object)</code>	Get object key names	<code>\$keys(response.body)</code>	<code>["id", "name"]</code>

Numeric (7)

Function	Description	Example	Result
<code>\$sum(array)</code>	Sum of array	<code>\$sum([1, 2, 3])</code>	6
<code>\$max(array)</code>	Maximum value	<code>\$max([10, 5, 20])</code>	20
<code>\$min(array)</code>	Minimum value	<code>\$min([10, 5, 20])</code>	5
<code>\$average(array)</code>	Arithmetic mean	<code>\$average([10, 20, 30])</code>	20
<code>\$abs(number)</code>	Absolute value	<code>\$abs(-7)</code>	7
<code>\$floor(number)</code>	Round down	<code>\$floor(3.9)</code>	3
<code>\$round(number, precision)</code>	Round to precision	<code>\$round(3.456, 2)</code>	3.46

String (7)

Function	Description	Example	Result
<code>\$uppercase(string)</code>	To uppercase	<code>\$uppercase('hello')</code>	"HELLO"

Function	Description	Example	Result
<code>\$lowercase(string)</code>	To lowercase	<code>\$lowercase('Hello')</code>	"hello"
<code>\$trim(string)</code>	Remove leading/trailing whitespace	<code>\$trim(' hi ')</code>	"hi"
<code>\$substring(string, start, length)</code>	Extract substring (zero-based)	<code>\$substring('abcdef', 2, 3)</code>	"cde"
<code>\$contains(string, pattern)</code>	Check if string contains pattern	<code>\$contains(session.user_agent, 'CTV')</code>	true or false
<code>\$match(string, pattern)</code>	Match string against regex pattern	<code>\$match('abc-123', /[0-9]+/)</code>	{"match": "123", ...}
<code>\$replace(string, pattern, replacement)</code>	Replace matching pattern	<code>\$replace('hello', 'l', 'r')</code>	"herro"

Array (5)

Function	Description	Example	Result
<code>\$append(arr1, arr2)</code>	Concatenate arrays	<code>\$append([1, 2], [3, 4])</code>	[1, 2, 3, 4]
<code>\$reverse(array)</code>	Reverse order	<code>\$reverse([1, 2, 3])</code>	[3, 2, 1]
<code>\$sort(array)</code>	Sort array	<code>\$sort([3, 1, 2])</code>	[1, 2, 3]

Function	Description	Example	Result
<code>\$distinct(array)</code>	Remove duplicates	<code>\$distinct([1, 2, 2, 3])</code>	<code>[1, 2, 3]</code>
<code>\$map(array, func)</code>	Apply function to each element	<code>\$map([1,2,3], function(\$v) {\$v*2})</code>	<code>[2, 4, 6]</code>

Boolean (1)

Function	Description	Example	Result
<code>\$not(value)</code>	Logical NOT	<code>\$not(false)</code>	<code>true</code>

Random (1)

Function	Description	Example	Result
<code>\$random()</code>	Random number between 0 (inclusive) and 1 (exclusive)	<code>\$random() > 0.5 ? 'A' : 'B'</code>	<code>"A" or "B"</code>

Note

`$random()` generates a new value on each evaluation. If you need the same random value in multiple output keys, bind it to a variable first: (`$r := $random(); ...`).

Date/time (4)

Function	Description	Example	Result
<code>\$now()</code>	Current timestamp as an ISO 8601 string	<code>\$now()</code>	"2024-01-15T12:00:00.000Z"
<code>\$millis()</code>	Current timestamp as milliseconds since epoch	<code>\$millis()</code>	1705320000000
<code>\$toMillis(string)</code>	Convert ISO 8601 string to milliseconds	<code>\$toMillis('2024-01-15T12:00:00.000Z')</code>	1705320000000
<code>\$fromMillis(number)</code>	Convert milliseconds to ISO 8601 string	<code>\$fromMillis(1705320000000)</code>	"2024-01-15T12:00:00.000Z"

Encoding (6)

Function	Description	Example
<code>\$encodeURIComponent(string)</code>	URL encode (preserves structural characters like /, ?, &)	<code>\$encodeURIComponent('https://example.com/path?q=hello world')</code>
<code>\$encodeURIComponentComponent(string)</code>	URL encode a single component (encodes all special characters)	<code>\$encodeURIComponent('a&b=c')</code> → "a%26b%3Dc"
<code>\$decodeUrl(string)</code>	Decode a URL-encoded string	<code>\$decodeUrl('hello%20world')</code> → "hello world"

Function	Description	Example
<code>\$decodeURIComponent(string)</code>	Decode a URL-encoded component	<code>\$decodeURIComponent('a%26b') → "a&b"</code>
<code>\$base64encode(string)</code>	Encode to Base64	<code>\$base64encode('hello') → "aGVsbG8="</code>
<code>\$base64decode(string)</code>	Decode from Base64	<code>\$base64decode('aGVsbG8=') → "hello"</code>

Tip

Use `$encodeURIComponent()` for individual query parameter values. Use `$encodeUrl()` only when you need to encode a full URL while preserving its structure.

Common patterns

Fallback values

Provide a default when a value might not exist.

```
{%$exists(player_params.region) ? player_params.region : 'us-east-1%'}
```

Dynamic URL construction

Build an ad decision server URL from multiple inputs.

```
{%'https://ads.example.com/v1/vast?ip=' & $encodeURIComponent(session.client_ip) &
'&ua=' & $encodeURIComponent(session.user_agent) & '&sid=' & session.id%}
```

Status code checks for HTTP_REQUEST output

Guard output values against HTTP failures.

```
{%response.statusCode != null and response.statusCode = 200 ? response.body.envelope :
'default-envelope%'}
```

Numeric conversion from player parameters

Player parameters arrive as strings. Convert them before arithmetic or numeric comparisons.

```
{%$number(player_params.max_duration) > 30 ? 'long' : 'short'%}
```

Important

If `$number()` receives a non-numeric string, it returns undefined. Combine it with `$exists()` when the parameter might be missing or invalid: (`$val := $number(player_params.max_duration); $exists($val) and $val > 30 ? 'long' : 'short'`).

Random traffic split

Assign viewers to experiment groups using `$random()`.

```
{%$random() > 0.5 ? 'https://ads.example.com/v1/vast-a' : 'https://ads.example.com/v1/vast-b'%}
```

Device type classification

Classify devices based on the user agent string.

```
{%$contains(session.user_agent, 'CTV') ? 'ctv' : $contains(session.user_agent, 'Mobile') ? 'mobile' : 'desktop'%}
```

Function examples

Before working through these examples, you should be familiar with:

- [Lifecycle hooks](#) — Input fields and output namespaces available at each lifecycle hook.
- [Function types and composition](#) — How each function type works and its configuration fields.
- [Creating and managing](#) — How to create functions and attach them to playback configurations.

This page provides complete, working function configurations for common use cases. Each example includes the scenario, the full configuration, the function mapping, and an explanation of what happens when the function runs.

Example	Scenario description
the section called “Example 1: Data enrichment”	Fetch a LiveRamp identity envelope at session start and store it in player parameters.
the section called “Example 2: A/B traffic split”	Split ad request traffic evenly between two ad decision server URLs for A/B testing.

Example 1: Data enrichment

Scenario

A streaming service wants to fetch audience identity data from an external service (in this case, LiveRamp) at session start and store it in player parameters for ad targeting later in the session. An identity envelope is an encrypted identifier provided by an identity resolution service that allows ad systems to recognize a viewer across devices and platforms without exposing personal information.

Configuration

Fetch LiveRamp envelope (HTTP_REQUEST):

```
{
  "FunctionId": "fetchLiveRamp",
  "FunctionType": "HTTP_REQUEST",
  "HttpRequestConfiguration": {
    "Runtime": "JSONATA",
    "MethodType": "GET",
    "Url": "{%'https://api.example-identity.com/v2/envelope?pid=12345&iv=' &
player_params.hashId%}",
    "Headers": {
      "Authorization": "{%'Bearer ramp-token-789'%}"
    },
    "RequestTimeoutMilliseconds": 2000,
    "Output": {
      "player_params.envelope_id": "{%response.body.envelopes[0].value%}"
    }
  }
}
```

Note

Replace `Bearer ramp-token-789` with your own API credentials. Do not store sensitive tokens directly in function configurations in production. Consider using a token rotation strategy.

Set `RequestTimeoutMilliseconds` based on the expected response time of your external service. A lower value reduces latency but increases the chance of timeout errors.

Function mapping

```
{
  "FunctionMapping": {
    "PRE_SESSION_INITIALIZATION": "fetchLiveRamp"
  }
}
```

What happens when the function runs

1. A viewer starts a playback session.
2. MediaTailor runs the `PRE_SESSION_INITIALIZATION` lifecycle hook and runs `fetchLiveRamp`.
3. The function builds the request URL using `player_params.hashedExceptionId` and calls the LiveRamp API. In the URL, `pid` is the partner ID and `iv` is the hashed viewer identifier.
4. The output expression extracts the value from the first envelope in the response array (`response.body.envelopes[0].value`) and writes it to `player_params.envelope_id`.

Tip

To handle errors explicitly, check `response.statusCode` before accessing response data:
`{%response.statusCode = 200 ? response.body.envelopes[0].value : ''%}`

For more information, see [HTTP_REQUEST](#), [PRE_SESSION_INITIALIZATION](#), [Troubleshooting and monitoring](#), and [Limits](#).

Example 2: A/B traffic split

Scenario

A streaming service wants to use a CUSTOM_OUTPUT function to randomly split ad request traffic between two ad decision server (ADS) URLs for A/B testing. Half of the ad requests go to the v1 endpoint and half go to the v2 endpoint.

Configuration

A/B traffic split (CUSTOM_OUTPUT):

```
{
  "FunctionId": "abTestAdsUrl",
  "FunctionType": "CUSTOM_OUTPUT",
  "CustomOutputConfiguration": {
    "Runtime": "JSONATA",
    "Output": {
      "adsRequest.url": "{%$random() < 0.5 ? 'https://ads.example.com/v1/decision?session=' & session.id : 'https://ads.example.com/v2/decision?session=' & session.id%}"
    }
  }
}
```

In this configuration:

- **FunctionId** — A unique name for this function.
- **FunctionType** — CUSTOM_OUTPUT evaluates expressions without making HTTP calls.
- **Output** — Maps `adsRequest.url` to an expression that randomly selects one of two ADS URLs.

Function mapping

```
{
  "FunctionMapping": {
    "PRE_ADS_REQUEST": "abTestAdsUrl"
  }
}
```

What happens when the function runs

1. An ad break is encountered during playback.
2. MediaTailor runs the `PRE_ADS_REQUEST` lifecycle hook and runs `abTestAdsUrl`.
3. The `$random()` function returns a value between 0 and 1. If the value is less than 0.5, the function sets the ADS URL to the v1 endpoint. Otherwise, it sets the URL to the v2 endpoint.

The `$random()` function is evaluated on each ad break, so the split is per ad request, not per session.

Tip

To adjust the traffic split ratio, change the threshold value. For example, `$random() < 0.8` sends 80% of traffic to the first URL and 20% to the second.

Note

Writing to `adsRequest.url` overrides the default ADS URL configured in the playback configuration for the current ad break.

For more information, see [CUSTOM_OUTPUT](#), [PRE_ADS_REQUEST](#), and [JSONata expression reference](#).

Troubleshooting and monitoring Functions

This page helps you diagnose common function errors and monitor function performance in production. The troubleshooting section is organized by symptom — start with what you observe, then follow the cause and fix.

Monitoring

CloudWatch metrics

MediaTailor publishes metrics for function execution to Amazon CloudWatch. No opt-in is required.

Hook-level metrics — one data point per lifecycle hook execution:

Metric	Description	Dimensions
PreSessionInitHook. .Invocations	Count of hook executions	ConfigurationName
PreSessionInitHook. .Errors	Count of hook errors	ConfigurationName
PreSessionInitHook. .Latency	Hook execution time (ms)	ConfigurationName
PreAdsRequestHook. .Invocations	Count of hook executions	ConfigurationName
PreAdsRequestHook. .Errors	Count of hook errors	ConfigurationName
PreAdsRequestHook. .Latency	Hook execution time (ms)	ConfigurationName

Function-level metrics — one data point per individual function execution:

Metric	Description	Dimensions
Function.Invocations	Count of function executions	ConfigurationName , FunctionId , FunctionT ype , HookType
Function.Errors	Count of function errors	ConfigurationName , FunctionId , FunctionT ype , HookType
Function.Latency	Function execution time (ms)	ConfigurationName , FunctionId , FunctionT ype , HookType

For more details on setting up alarms and working with these metrics, see [Monitoring AWS Elemental MediaTailor with Amazon CloudWatch metrics](#).

Log events

MediaTailor emits log events for function execution. Error events are emitted by default. Completed and summary events are opt-in.

Event type	Default / Opt-in	Log group	Description
PRE_SESSION_INIT_HOOK_SUMMARY	Opt-in	Manifest Log	Hook execution summary (success/error)
PRE_SESSION_INIT_HOOK_ERROR	Default	Manifest Log	Hook failure with errorType and cause
PRE_SESSION_INIT_FUNCTION_COMPLETED	Opt-in	Manifest Log	Individual function completed with input/output
PRE_SESSION_INIT_FUNCTION_ERROR	Default	Manifest Log	Individual function failure
PRE_ADS_REQUEST_HOOK_SUMMARY	Opt-in	ADS Interaction Log	Hook execution summary (success/error)
PRE_ADS_REQUEST_HOOK_ERROR	Default	ADS Interaction Log	Hook failure with errorType and cause
PRE_ADS_REQUEST_FUNCTION_COMPLETED	Opt-in	ADS Interaction Log	Individual function completed with input/output

Event type	Default / Opt-in	Log group	Description
PRE_ADS_R EQUEST_FU NCTION_ERROR	Default	ADS Interaction Log	Individual function failure

To enable opt-in log events, see [Monitoring AWS Elemental MediaTailor with Amazon CloudWatch metrics](#).

Use the `eventId` field to correlate hook-level and function-level events for the same execution.

The following Amazon CloudWatch Logs Insights query filters function error events by `eventId` to trace a single execution:

```
fields @timestamp, eventType, functionId, errorType, cause
| filter eventId = "5dc6f040-0f72-4e8c-a64e-25eeef62708c"
| sort @timestamp asc
```

Troubleshooting

When a function fails, MediaTailor logs an `errorType` field in the error event. Use this field to identify the class of failure:

Error type	Description
SYNTAX_ERROR	Expression failed to compile or encountered a runtime type error
RESOURCE_LIMIT_ERROR	Expression exceeded CPU time, memory, or stack depth limits
RESTRICTION_ERROR	Expression used a blocked function, or input payload size limit exceeded
TIMEOUT_ERROR	Function execution exceeded its time limit
VALIDATION_ERROR	Output path targets a field not writable in the current hook's scope

Error type	Description
INTERNAL_ERROR	Infrastructure failure unrelated to the function

The entries are organized by symptom and reference these error types where applicable.

Expression returns null unexpectedly

Symptom: An output value you expected to be populated is `null` or missing from player parameters.

Possible causes:

Cause	How to identify	Fix
The input field does not exist at this lifecycle hook.	You referenced <code>adsRequest.url</code> in a <code>PRE_SESSION_INITIALIZATION</code> function. ADS request data is not available at session start.	Move the function to the <code>PRE_ADS_REQUEST</code> lifecycle hook, or use a different input field. See Lifecycle hooks .
The input field is missing from the session data.	You referenced <code>player_params.campaign_id</code> , but the player did not pass that parameter at session initialization.	Use <code>\$exists()</code> to check before accessing: <code>{%\$exists(player_params.campaign_id) ? player_params.campaign_id : 'default'%}</code> .
You wrote an object or array to player parameters or the ADS request.	These namespaces accept only strings, numbers, and booleans. Objects and arrays are filtered out.	Store complex data in <code>temp.*</code> and extract strings, numbers, or booleans in a subsequent step.

RESOURCE_LIMIT_ERROR: Stack overflow

Symptom: The function fails with `errorType: "RESOURCE_LIMIT_ERROR"` and `cause: "Stack overflow error"`.

Cause: The expression exceeded the maximum stack depth of 100 levels. This typically happens with deeply nested conditional (if/then/else) expressions or complex variable assignments.

This means the expression has too many levels of nesting for MediaTailor to process.

Fix: Simplify the expression. Break complex logic into multiple output entries or multiple steps in a sequential executor.

RESOURCE_LIMIT_ERROR: CPU timeout

Symptom: The function fails with `errorType: "RESOURCE_LIMIT_ERROR"` and `cause: "Expression evaluation timeout: Check for infinite loop"`.

Cause: The expression exceeded the 100 ms CPU time limit. This can happen with expressions that perform expensive computations over large data structures.

Fix: Reduce the complexity of the expression. If you are processing large arrays, consider moving that logic to an external service and calling it with an `HTTP_REQUEST` function.

RESTRICTION_ERROR: Function not allowed

Symptom: The function fails with `errorType: "RESTRICTION_ERROR"` and `cause: "Function '<name>' is not allowed"`.

Cause: The expression calls a JSONata function that is not in the allowed list of 38 functions. Common examples include `$filter`, `$reduce`, `$eval`, `$split`, and `$join`.

Fix: Check the `cause` field for the blocked function name. Replace it with an allowed alternative. See [JSONata expression reference](#) for the full list of 38 allowed functions.

Commonly used allowed functions include `$string`, `$number`, `$substring`, `$contains`, and `$encodeURIComponent`.

RESTRICTION_ERROR: Expression too long

Symptom: The function fails to create or update with `cause: "Expression length <actual> exceeds limit <limit>"`.

Cause: A single expression exceeds 1,000 characters.

Fix: Break the expression into smaller parts. Use multiple output entries, or split the logic across multiple steps in a sequential executor. Use variable binding (`: =`) to avoid repeating long subexpressions.

HTTP error: Status code is null

Symptom: In an `HTTP_REQUEST` function's output, `response.statusCode` is `null`.

Cause: The external API was unreachable, the connection timed out, or a network error occurred. When this happens, MediaTailor sets `response.statusCode` to `null`, `response.body` to `null`, and `response.text` to "Internal Error".

Fix: Always check `response.statusCode` before accessing response data:

```
{%response.statusCode = 200 ? response.body.value : 'default'%}
```

This expression checks whether the HTTP call returned a 200 status code. If it did, it uses the response value. Otherwise, it falls back to a default value.

If this happens frequently, check whether the external API is healthy. Consider increasing `RequestTimeoutMilliseconds` if the API is slow, or decreasing it if you want to fail fast.

HTTP error: Response body is null

Symptom: `response.statusCode` is 200 but `response.body` is `null`.

Cause: The response body is either not valid JSON or exceeds 20,000 characters. MediaTailor only parses JSON responses up to 20,000 characters into `response.body`.

Fix: Use `response.text` as a fallback. The `response.text` field contains the raw response body truncated to 20,000 characters:

```
{%response.statusCode = 200 ? ($exists(response.body.id) ? response.body.id :  
  $substring(response.text, 0, 100)) : 'error'%}
```

If the data you need is beyond the 20,000-character limit, consider asking the external API to return a smaller response (for example, by requesting specific fields).

HTTP error: URL validation failure

Symptom: The function fails at runtime with a message about the URL being malformed, using an invalid scheme, or exceeding the maximum length.

Possible causes:

Cause	Fix
The URL does not use http or https.	Ensure the URL expression produces a URL starting with <code>http://</code> or <code>https://</code> .
The URL exceeds 2,048 characters after expression evaluation.	Shorten the URL. Move large parameter values to the request body using a POST method.
The URL is malformed (not a valid URI).	Check the expression for missing or extra characters. Use <code>\$encodeURIComponent()</code> for query parameter values that may contain special characters.

VALIDATION_ERROR

Symptom: The function fails with `errorType: "VALIDATION_ERROR"`. This error can occur at authoring time (when you create or update the function) or at execution time (when the function runs during a session).

Possible causes:

Cause	Example	Fix
Output key targets a namespace not writable at the current hook.	Writing to <code>adsRequest.url</code> in a function attached to <code>PRE_SESSION_INITIALIZATION</code> .	Check which output namespaces are allowed at your lifecycle hook. <code>PRE_SESSION_INITIALIZATION</code> only allows <code>player_params.*</code> . Either move the function to the correct hook or change the output key. See Lifecycle hooks .
Output key uses invalid characters.	An output key like <code>player_params.device type</code> (with a space). Only	Rename the output key to use only valid characters. For example, use <code>player_pa</code>

Cause	Example	Fix
	letters, numbers, underscores, and hyphens are allowed.	<code>rams.device_type</code> instead.
Output key doesn't start with a valid prefix.	An output key like <code>custom.myValue</code> instead of <code>player_params.myValue</code> or <code>temp.myValue</code> .	Use a valid output prefix: <code>player_params.*</code> , <code>temp.*</code> , or <code>adsRequest.*</code> .
A JSONata expression has a syntax error.	A missing closing quote or incomplete conditional: <code>{%session.id & %}</code> .	Review the expression for missing quotes, unmatched parentheses, or unsupported operators like <code>??</code> or <code>?:</code> .
A required field is missing in an <code>HTTP_REQUEST</code> function.	The URL field is empty or the method is not specified.	Ensure the URL and method fields are set. The method must be GET or POST.
The URL built by the expression is invalid.	The evaluated URL uses an unsupported scheme like <code>ftp://</code> , exceeds 2,048 characters, or is malformed.	Verify the URL expression produces a valid <code>http://</code> or <code>https://</code> URL. Use <code>\$encodeURIComponent()</code> for query parameter values that may contain special characters.
An HTTP header contains invalid characters or uses a restricted name.	A header value contains line breaks, or the header name is <code>host</code> or <code>transfer-encoding</code> .	Remove invalid characters from header values. Avoid restricted header names. See HTTP_REQUEST for header limits.

Check the cause field in the error log event — it identifies which field or expression failed validation.

INTERNAL_ERROR

Symptom: The function fails with `errorType`: "INTERNAL_ERROR".

Cause: An infrastructure failure occurred that is unrelated to your function configuration.

Fix: Retry the request. If the error persists, contact AWS Support.

Functions limits

This page lists the limits that apply to Functions. These limits are enforced at authoring time (when you create or update a function) and at runtime (when the function runs).

Hook execution limits

Hook	Limit	Value
PRE_SESSION_INITIALIZATION	Total timeout	2,000 ms
PRE_SESSION_INITIALIZATION	Total <code>player_params</code> size	1,000 characters
PRE_ADS_REQUEST	Total timeout	2,000 ms

These timeouts cover the entire hook execution, including all function steps and HTTP calls. If the hook exceeds its timeout, MediaTailor discards all output and proceeds as if no function were attached.

Individual function timeouts (such as `RequestTimeoutMilliseconds` for `HTTP_REQUEST` functions) must fit within the hook's total timeout. For example, if the hook timeout is 2,000 ms and an `HTTP_REQUEST` function sets `RequestTimeoutMilliseconds` to 2,000 ms, the function may time out before completing if any processing occurs before or after the HTTP call.

The `player_params` total size includes the combined character count of all output key names and values. For example, outputting `player_params.deviceType = "mobile"` and `player_params.region = "us-east-1"` counts `deviceType`, `mobile`, `region`, and `us-east-1` toward the 1,000-character total.

Expression limits

Limit	Value	Enforced at
CPU time per expression	100 ms	Runtime
Stack depth	100	Runtime
Expression length	1,000 characters	Authoring

When an expression exceeds a limit at runtime, MediaTailor stops the expression and records the error. For details on specific error types, see [Troubleshooting and monitoring](#).

Function composition limits

Limit	Value
Steps per sequential executor	1–10
Maximum nesting depth	2
Total function executions per hook	20
Output entries per function	20
Circular references	Not allowed

These limits are enforced at authoring time.

HTTP request limits

Limit	Value
URL length	2,048 characters
Request body size	64 KB
Header count	50

Limit	Value
Header name length	256 characters
Header value length	8,192 characters
Request timeout	100–2,000 ms (customer-configurable via <code>RequestTimeoutMilliseconds</code>)
Allowed URL schemes	https, http
Restricted headers	Host, Transfer-Encoding , Content-Length , Connection

URL length, body size, header count, and header size limits are enforced at authoring time. Request timeout and restricted headers are enforced at runtime.

If a function sets a restricted header, MediaTailor accepts the function configuration at authoring time but drops the header when the HTTP request is sent during execution.

Response limits

Limit	Value
<code>response.body</code> maximum size	20,000 characters
<code>response.text</code> maximum length	20,000 characters

These limits are enforced at runtime.

Output limits

Limit	Value
Output value max length (per key)	1,000 characters

These limits are enforced at runtime.

Allowed JSONata functions

For the full list of allowed functions, see [JSONata expression reference](#).

For help diagnosing errors related to these limits, see [Troubleshooting and monitoring](#).

MediaTailor dynamic ad variables for ADS requests

AWS Elemental MediaTailor uses dynamic ad variables to pass information from your viewing session to the ad decision server (ADS). This information helps the ADS select the most relevant ads for your viewers.

This section provides an overview of dynamic ad variables and links to specific implementation guides. For step-by-step configuration instructions, see the individual topics below.

Dynamic variable types

MediaTailor supports four types of dynamic variables:

- **Session variables** – Automatically generated values like session ID and SCTE-35 data. See [MediaTailor session variables for ADS requests](#).
- **Player variables** – Custom parameters sent by your video player. See [MediaTailor player variables for ADS requests](#).
- **Domain variables with Configuration aliases** – Dynamic URL domains for multi-origin configurations.
- **Configuration aliases** – Predefined mappings for dynamic variable replacement. See [Configuration aliases](#).

Common use cases

Use dynamic ad variables to:

- Pass viewer demographics and preferences to your ADS
- Route requests to different origins based on geographic location
- Enable time-shifted viewing with MediaPackage integration
- Implement A/B testing and failover scenarios

The following sections provide additional detail about using dynamic ad variables with MediaTailor.

Topics

- [MediaTailor session variables for ADS requests](#)
- [MediaTailor player variables for ADS requests](#)
- [MediaTailor domain variables for multiple content sources](#)
- [MediaTailor configuration aliases overview](#)
- [MediaTailor passing parameters to ADS](#)
- [MediaTailor parameter routing for ADS and origins](#)
- [MediaTailor and MediaPackage time-shifted viewing integration](#)
- [MediaTailor parameter session behavior and persistence](#)
- [MediaTailor parameter reference and limitations](#)
- [MediaTailor parameter troubleshooting guide](#)
- [MediaTailor configuration aliases troubleshooting guide](#)

For parameter formatting requirements and troubleshooting, see [MediaTailor parameter reference and limitations](#) and [MediaTailor parameter troubleshooting guide](#).

MediaTailor session variables for ADS requests

AWS Elemental MediaTailor sends session data to the Ad Decision Server (ADS) when you configure AWS Elemental MediaTailor to specify one or more of the variables listed in this section in the template ADS URL. You can use individual variables, and you can concatenate multiple variables to create a single value. MediaTailor generates some values and obtains the rest from sources like the manifest and the player's session initialization request.

The following table describes the session data variables you can use in your template ADS request URL configuration. The section numbers listed in the table correspond to the 2019a version of the Society of Cable Telecommunications Engineers (SCTE)-35 specification, [Digital Program Insertion Cueing Message](#). For details about ad prefetch, see [Prefetching ads](#).

Name	Available for ad prefetch	SCTE-35 specification section	Description
[avail.index]	Yes		A number that represents the position of an ad avail in an index. At the start of a playback session, MediaTailor creates an index of all the ad avails in a manifest and stores the index for the remainder of the session. When MediaTailor makes a request to the ADS to fill the avail, it includes the ad avail index number. This parameter enables the ADS to improve ad selection by using features like competitive exclusion and frequency capping.
[avail.random]	Yes		A random number between 0 and 10,000,000,000, as a long number, that MediaTailor generates for each request to the ADS. Some ad servers use this parameter to enable features such as separating ads from competing companies.
[scte.archive_allowed_flag]	Yes	10.3.3.1	An optional Boolean value. When this value is 0, recording restrictions are asserted on the segment. When this value is 1, recording restrictions are not asserted on the segment.
[scte.avail_num]	Yes	9.7.2.1	The value parsed by MediaTailor from the SCTE-35 field <code>avail_num</code> , as a long number. MediaTailor can use this value to designate linear ad avail numbers. Value must be an integer.
[scte.avails_expected]	Yes	9.7.2.1	An optional long value that gives the expected count of avails within the current event.
[scte.delivery_not]	Yes	10.3.3.1	An optional Boolean value. When this value is 0, the next five bits are reserved. When this value is 1, the

Name	Available for ad prefetch	SCTE-35 specification section	Description
_restricted_flag]			next five bits take on the meanings as described in the SCTE-35 specification.
[scte.device_restrictions]	Yes	10.3.3.1	An optional integer value that signals three pre-defined, independent, and non-hierarchical groups of devices. For more information about this variable, see the segments_expected description in the SCTE-35 specification.
[scte.event_id]	Yes	9.1 and 9.7.2.1	The value parsed by MediaTailor from the SCTE-35 field <code>splice_event_id</code> , as a long number. MediaTailor uses this value to designate linear ad avail numbers or to populate ad server query strings, like ad pod positions. Value must be an integer.
[scte.no_regional_blackout_flag]	Yes	10.3.3.1	An optional Boolean value. When this value is 0, regional blackout restrictions apply to the segment. When this value is 1, regional blackout restrictions do not apply to the segment.
[scte.segment_num]	Yes	10.3.3.1	An optional integer value that numbers segments within a collection of segments. For more information about this variable, see the segment_num description in the SCTE-35 specification.
[scte.segmentation_event_id]	Yes	10.3.3.1	MediaTailor exposes this variable as scte.event_id .

Name	Available for ad prefetch	SCTE-35 specification section	Description
[scte.segmentation_type_id]	Yes	10.3.3.1	An optional 8-bit integer value that specifies the segmentation type. For more information about this variable, see the segmentation_type_id description in the SCTE-35 specification.

Name	Available for ad prefetch	SCTE-35 specification section	Description
[scte.segmentation_upid]	segmentation_upid_type : Yes private_data : Yes	segmentation_upid: 10.3.3.1 Managed Private UPID: 10.3.3.3	Corresponds to the SCTE-35 <code>segmentation_upid</code> element. The <code>segmentation_upid</code> element contains <code>segmentation_upid_type</code> and <code>segmentation_upid_length</code> . MediaTailor supports the following <code>segmentation_upid</code> types: • ADS Information (0x0E) - Advertising information. For more information, see the segmentation_upid description in the SCTE-35 specification. • Managed Private UPID (0x0C) - The Managed Private UPID (MPU) structure as defined in the SCTE-35 specification. MediaTailor supports binary or DASH XML SCTE representations. You can use this structure in a podbustler workflow. To do so, specify a 32-bit (4 byte) <code>format_id</code> identifier , and include the following parameters in the <code>private_data</code> attribute: <pre>ABCD{"assetId": " my_program ", "cueData": {"cueType": " theAdType ", "key": " pb", "value": " 123456"}}</pre> MediaTailor parses the values from the preceding JSON, and passes them into the <code>scte.segmentation_upid.assetId</code> , <code>scte.segmentation_upid.cueData.key</code> , and <code>scte.segmentation_upid.cueData.value</code> dynamic variables.

Name	Available for ad prefetch	SCTE-35 specification section	Description
			• User Defined (0x01) - A user defined structure. For more information, see the segmentation_upid description in the SCTE-35 specification.
<code>[scte.segmentation_upid.assetId]</code>	Yes		Used in conjunction with the Managed Private UPID (0xC) <code>segmentation_upid_type</code> for podbustor workflows. MediaTailor derives this value from the <code>assetId</code> parameter in the MPU's <code>private_data</code> JSON structure. For more information, see Managed Private UPID JSON structure for a podbustor workflow .
<code>[scte.segmentation_upid.cueData.key]</code>	Yes		Used in conjunction with the Managed Private UPID (0xC) <code>segmentation_upid_type</code> for podbustor workflows. MediaTailor derives this value from the <code>cueData.key</code> parameter in the MPU's <code>private_data</code> JSON structure. For more information, see Managed Private UPID JSON structure for a podbustor workflow .
<code>[scte.segmentation_upid.cueData.value]</code>	Yes		Used in conjunction with the Managed Private UPID (0xC) <code>segmentation_upid_type</code> for podbustor workflows. MediaTailor derives this value from the <code>cueData.key</code> parameter in the MPU's <code>private_data</code> JSON structure. For more information, see Managed Private UPID JSON structure for a podbustor workflow. Value can be a string.

Name	Available for ad prefetch	SCTE-35 specification section	Description
<code>[scte.segmentation_upid.private_data.{index}]</code>	Yes		Used in conjunction with the Managed Private UPID (0xC) segmentation_upid_type for targeted advertising workflows. MediaTailor splits colon-delimited segmentation UPID tokens and creates indexed session variables. The index corresponds to the position in the colon-delimited list, ignoring leading whitespace from the initial colon. For example, if segmentation_upid = ":3213214:2313321/5:3943" , then: <code>[scte.segmentation_upid.private_data.0]</code> = 3213214 <code>[scte.segmentation_upid.private_data.1]</code> = 2313321/5 <code>[scte.segmentation_upid.private_data.2]</code> = 3943 Value can be a string.
<code>[scte.segments_expected]</code>	Yes	10.3.3.1	An optional integer value that gives the expected count of individual segments within a collection of segments. For more information about this variable, see the segments_expected description in the SCTE-35 specification.
<code>[scte.sub_segment_num]</code>	Yes	10.3.3.1	An optional integer value that identifies a particular sub-segment within a collection of sub-segments. For more information about this variable, see the sub_segment_num description in the SCTE-35 specification.

Name	Available for ad prefetch	SCTE-35 specification section	Description
<code>[scte.sub_segments_expected]</code>	Yes	10.3.3.1	An optional integer value that gives the expected count of individual sub-segments within a collection of sub-segments. For more information about this variable, see the sub_segments_expected description in the SCTE-35 specification.
<code>[scte.unique_program_id]</code>	Yes	9.7.2.1	The integer value parsed by MediaTailor from the SCTE-35 <code>splice_insert</code> field <code>unique_program_id</code>. The ADS uses the unique program ID (UPID) to provide program-level ad targeting for live linear streams. If the SCTE-35 command is not <code>splice insert</code>, MediaTailor sets this to an empty value. Value must be an integer.

Name	Available for ad prefetch	SCTE-35 specification section	Description
[session.avail_duration_ms]	Yes		The duration in milliseconds of the ad availability slot. The default value is 300,000 ms. AWS Elemental MediaTailor obtains the duration value from the input manifest as follows: - For HLS: MediaTailor obtains the duration from the <code>#EXT-X-CUE-OUT: DURATION</code> or from values in the <code>#EXT-X-DATERANGE</code> tag. If the input manifest has a null, invalid, or 0 duration for the ad avail in those tags, MediaTailor uses the default. - For DASH: MediaTailor obtains the duration value from the event duration, if one is specified. Otherwise, it uses the default value. - For VOD: when a VOD stream triggers a pre-roll ad call, if the manifest does not include SCTE messaging with a duration value, MediaTailor does not enter a duration for the <code>[session.avail_duration_ms]</code>, including the default duration value.
[session.avail_duration_secs]	Yes		The duration in seconds of the ad availability slot, or ad avail, rounded to the nearest second. MediaTailor determines this value the same way it determines <code>[session.avail_duration_ms]</code>.
[session.client_ip]	No		The remote IP address that the MediaTailor request came from. If the <code>X-forwarded-for</code> header is set, then that value is what MediaTailor uses for the <code>client_ip</code>.

Name	Available for ad prefetch	SCTE-35 specification section	Description
[session.id]	No		A unique numeric identifier for the current playback session. All requests that a player makes for a session have the same id, so it can be used for ADS fields that are intended to correlate requests for a single viewing.
[session.referrer]	No		Usually, the URL of the page that is hosting the video player. MediaTailor sets this variable to the value of the <code>Referer</code> header that the player used in its request to MediaTailor. If the player doesn't provide this header, MediaTailor leaves the [session.referrer] empty. If you use a content delivery network (CDN) or proxy in front of the manifest endpoint and you want this variable to appear, proxy the correct header from the player here.
[session.user_agent]	No		The <code>User-Agent</code> header that MediaTailor received from the player's session initialization request. If you're using a CDN or proxy in front of the manifest endpoint, you must proxy the correct header from the player here.
[session.uuid]	No		Alternative to [session.id] . This is a unique identifier for the current playback session, such as the following: e039fd39-09f0-46b2-aca9-9871cc116cde

Name	Available for ad prefetch	SCTE-35 specification section	Description
<code>[avail.source_content_time_epoch_ms]</code>	No		For HLS the value is the PDT of the origin segment that started the avail. For DASH the value is the <code><SupplementalProperty> urn:scte:dash:utc-time</code> of the <code><Period></code> that contains the <code><EventStream></code> . - For preroll HLS or DASH ads <code>[avail.source_content_time_epoch_ms]</code> is the PDT of the first HLS segment, and the <code><SupplementalProperty> urn:scte:dash:utc-time</code> of the first <code><Period></code>. For live streams with short manifest windows this value will be different for each viewer based on when they start watching the stream. For a livestream with a full event DVR window the value will be the same for each viewer. - For partial avails due to avail suppression, <code>[avail.source_content_time_epoch_ms]</code> is the PDT of the source content segment that started the avail. For example, if MediaTailor removes the first 20 seconds of the avail, <code>[avail.source_content_time_epoch_ms]</code> will still be set to the original PDT, not shifted by 20 seconds. - For prefetch ADS requests, <code>[avail.source_content_time_epoch_ms]</code> is empty, however customers can set it using prefetch schedule retrieval dynamic variables.

Example

If the ADS requires a query parameter named `deviceSession` to be passed with the unique session identifier, the template ADS URL in AWS Elemental MediaTailor could look like the following:

```
https://my.ads.server.com/path?deviceSession=[session.id]
```

AWS Elemental MediaTailor automatically generates a unique identifier for each stream, and enters the identifier in place of `session.id`. If the identifier is `1234567`, the final request that MediaTailor makes to the ADS would look something like this:

```
https://my.ads.server.com/path?deviceSession=1234567
```

If the ADS requires several query parameters to be passed, the template ADS URL in AWS Elemental MediaTailor could look like the following:

```
https://my.ads.server.com/sample?
e=[scte.avails_expected]&f=[scte.segment_num]&g=[scte.segments_expected]&h=[scte.sub_segment_num]
```

The following DASH marker example XML fragment shows how to use `scte35:SpliceInsert`:

```
<Period start="PT444806.040S" id="123456" duration="PT15.000S">
  <EventStream timescale="90000" schemeIdUri="urn:scte:scte35:2013:xml">
    <Event duration="1350000">
      <scte35:SpliceInfoSection protocolVersion="0" ptsAdjustment="180832" tier="4095">
        <scte35:SpliceInsert spliceEventId="1234567890"
spliceEventCancelIndicator="false" outOfNetworkIndicator="true"
spliceImmediateFlag="false" uniqueProgramId="1" availNum="1" availsExpected="1">
          <scte35:Program><scte35:SpliceTime ptsTime="5672624400"/></scte35:Program>
          <scte35:BreakDuration autoReturn="true" duration="1350000"/>
        </scte35:SpliceInsert>
      </scte35:SpliceInfoSection>
    </Event>
  </EventStream>
</Period>
```

The following DASH marker example XML fragment shows how to use `scte35:TimeSignal`:

```
<Period start="PT346530.250S" id="123456" duration="PT61.561S">
  <EventStream timescale="90000" schemeIdUri="urn:scte:scte35:2013:xml">
    <Event duration="5310000">
      <scte35:SpliceInfoSection protocolVersion="0" ptsAdjustment="183003" tier="4095">
        <scte35:TimeSignal time="5310000"/>
      </scte35:SpliceInfoSection>
    </Event>
  </EventStream>
</Period>
```

```

    <scte35:TimeSignal>
      <scte35:SpliceTime ptsTime="3442857000"/>
    </scte35:TimeSignal>
    <scte35:SegmentationDescriptor segmentationEventId="1234567"
segmentationEventCancelIndicator="false" segmentationDuration="8100000"
segmentationTypeId="52" segmentNum="0" segmentsExpected="0">
      <scte35:DeliveryRestrictions webDeliveryAllowedFlag="false"
noRegionalBlackoutFlag="false" archiveAllowedFlag="false" deviceRestrictions="3"/>
      <scte35:SegmentationUpid segmentationUpidType="12"
segmentationUpidLength="2">0100</scte35:SegmentationUpid>
    </scte35:SegmentationDescriptor>
  </scte35:SpliceInfoSection>
</Event>

```

The following DASH marker example XML fragment shows how to use `scte35:Binary`:

```

<Period start="PT444806.040S" id="123456" duration="PT15.000S">
  <EventStream schemeIdUri="urn:scte:scte35:2014:xml+bin" timescale="1">
    <Event presentationTime="1541436240" duration="24" id="29">
      <scte35:Signal xmlns="http://www.scte.org/schemas/35/2016">
        <scte35:Binary>/DAhAAAAAAAAAAP/wEAUAAAHaf+9/fgAg9YDAAAAAAAAA25aoh</Binary>
      </scte35:Signal>
    </Event>
    <Event presentationTime="1541436360" duration="24" id="30">
      <scte35:Signal xmlns="http://www.scte.org/schemas/35/2016">
        <scte35:Binary>QW5vdGhlciB0ZXN0IHN0cm1uZyBmb3IgaWZw5jb2RpbmcgdG8gQmFzZTY0IGVuY29kZWQgYmluYXJ5J5Lg=
Binary>
      </scte35:Signal>
    </Event>
  </EventStream>
</Period>

```

The following HLS tag example shows how to use EXT-X-DATERANGE:

```

#EXT-X-DATERANGE:ID="splice-6FFFFFF0",START-DATE="2014-03-05T11:
15:00Z",PLANNED-DURATION=59.993,SCTE35-OUT=0xFC002F0000000000FF0
00014056FFFFFF000E011622DCAFF000052636200000000000A0008029896F50
0000087000000000

```

The following HLS tag example shows how to use EXT-X-CUE-OUT:

```

#EXT-OATCLS-SCTE35:/DA0AAAAAAAAAAAAABQb+ADAQ6QAeAhxDVUVJQAAA03/PAAEUrEoICAAAAAag
+2UBNAAANvrttoQ==

```

```
#EXT-X-ASSET:CAID=0x0000000020FB6501
#EXT-X-CUE-OUT:201.467
```

The following HLS tag example shows how to use EXT-X-SPLICEPOINT-SCTE35:

```
#EXT-X-SPLICEPOINT-SCTE35:/DA9AAAAAAAAAAP/wBQb+uYbZqwAnAiVDVUVJAAAKqX//
AAEjW4AMEU1EU05CMDAXMTMyMjE5M190NAAAmXz5JA==
```

The following example shows how to use scte35:Binary decode:

```
{
  "table_id": 252,
  "section_syntax_indicator": false,
  "private_indicator": false,
  "section_length": 33,
  "protocol_version": 0,
  "encrypted_packet": false,
  "encryption_algorithm": 0,
  "pts_adjustment": 0,
  "cw_index": 0,
  "tier": "0xFFF",
  "splice_command_length": 16,
  "splice_command_type": 5,
  "splice_command": {
    "splice_event_id": 448,
    "splice_event_cancel_indicator": false,
    "out_of_network_indicator": true,
    "program_splice_flag": true,
    "duration_flag": true,
    "splice_immediate_flag": false,
    "utc_splice_time": {
      "time_specified_flag": false,
      "pts_time": null
    },
  },
  "component_count": 0,
  "components": null,
  "break_duration": {
    "auto_return": false,
    "duration": {
      "pts_time": 2160000,
      "wall_clock_seconds": 24.0,
      "wall_clock_time": "00:00:24:000000"
    }
  }
}
```



```
    },
    "unique_program_id": 49152,
    "avail_num": 0,
    "avails_expected": 0
    "segment_num": 0,
    "segments_expected": 0,
    "sub_segment_num": 0,
    "sub_segments_expected": 0
  },
  "splice_descriptor_loop_length": 0,
  "splice_descriptors": null,
  "Scte35Exception": {
    "parse_status": "SCTE-35 cue parsing completed with 0 errors.",
    "error_messages": [],
    "table_id": 252,
    "splice_command_type": 5
  }
}
```

MediaTailor player variables for ADS requests

AWS Elemental MediaTailor sends data received from the player to the ADS when you configure AWS Elemental MediaTailor to specify `player_params.<query_parameter_name>` variables in the template ADS URL. For example, if the player sends a query parameter named `user_id` in its request to MediaTailor, to pass that data in the ADS request, include `[player_params.user_id]` in the ADS URL configuration.

This allows you to control the query parameters that are included in the ADS request. Typically, you add a special query parameter that the ADS recognizes to the ADS request URL and provide key-value pairs as the value of the parameter.

The examples used in the following procedure use the following key-value pairs:

- *param1* with a value of *value1*:
- *param2* with a value of *value2*:

To add query parameters as key-value pairs

1. In AWS Elemental MediaTailor, configure the ADS request template URL to reference the parameters. The following URL shows the inclusion of the example parameters:

```
https://my.ads.com/path?param1=[player_params.param1]&param2=[player_params.param2]
```

2. (Optional) For server-side ad-tracking reporting, URL-encode the key-value pairs on the player. When MediaTailor receives the session initialization request, it URL-decodes the values once before substituting them into the ADS request URL.

 Note

If your ADS requires a URL-encoded value, URL-encode the value twice on the player. This way, the decoding done by MediaTailor results in a once-encoded value for the ADS.

For example, if the decoded representation of the values sent to the ADS is `param1=value1:&param2=value2:`, then the URL-encoded representation is `param1=value1%3A&param2=value2%3A`.

3. In the session initialization call from the player, pass the key-value pairs to MediaTailor as the value of a single query parameter. The following example calls provide the example key-value pairs for server- and client-side ad tracking reporting.

- Example requests for server-side ad-tracking reporting - using URL-encoded pairs

HLS:

```
<master>.m3u8?ads.param1=value1%3A&ads.param2=value2%3A
```

DASH:

```
<manifest>.mpd?ads.param1=value1%3A&ads.param2=value2%3A
```

- Example request for client-side ad-tracking reporting - with no URL-encoding

HLS:

```
POST <master>.m3u8
{
  "adsParams": {
    "param1": "value1:",
    "param2": "value2:"
  }
}
```

```
}
```

DASH:

```
POST <manifest>.mpd
{
  "adsParams": {
    "param1": "value1:",
    "param2": "value2:"
  }
}
```

For server-side reporting, MediaTailor decodes the parameters when the player request is received. For client-side reporting, it doesn't alter the parameters received in the JSON payload. MediaTailor sends the following request to the ADS:

```
https://my.ads.com/<path>?param1=value1:&param2=value2:
```

In this way, the `param1` and `param2` key-value pairs are included as first-class query parameters in the ADS request.

MediaTailor domain variables for multiple content sources

AWS Elemental MediaTailor dynamic domain variables allow you to use multiple domains, such as the **my-ads-server.com** part of the URL `http://my-ads-server.com`, with the player parameters in your configuration. This makes it possible for you to use more than one content source or ad decision server (ADS) in a single configuration.

You can use domain variables with any parameter that contains a URI:

- `AdDecisionServerUrl`
- `AdSegmentUrlPrefix`
- `ContentSegmentUrlPrefix`
- `LivePreroll.AdDecisionServerUrl`
- `VideoContentSourceUrl`

Domain variables are used alongside *configuration aliases* to perform dynamic variable replacement. Configuration aliases map a set of aliases and values to the player parameters that are used for dynamic domain configuration. For setup procedures, see [Creating and using configuration aliases with MediaTailor](#). For detailed reference information, see [MediaTailor configuration aliases overview](#).

MediaTailor configuration aliases overview

AWS Elemental MediaTailor configuration aliases enable dynamic variable replacement in URL domains and other supported fields. Use this feature to use multiple domains and dynamically configure URLs during session initialization.

Use cases

Configuration aliases enable sophisticated multi-configuration architectures for the following scenarios:

- **Geographic routing:** Route requests to different origins or ad servers based on viewer location using region-specific aliases. For implementation guidance, see [CloudFront Origin Failover](#).
- **Content-based routing:** Direct different content types to specialized origins or processing pipelines. For routing behavior configuration, see [the section called “Set up CDN routing behaviors”](#).
- **Failover scenarios:** Implement backup origins and ad servers with automatic failover using alias switching. For detailed implementation, see [Implement multi-Region resilience for MediaTailor with MQAR](#) and [the section called “Plan CDN integration”](#).
- **A/B testing:** Test different ad servers, origins, or configurations by routing traffic based on player parameters. For load testing guidance, see [Prepare and run performance tests for Amazon CloudFront with real user monitoring](#).
- **Device-specific optimization:** Optimize content delivery and ad serving for different device types or capabilities. For comprehensive guidance, see [Set up manifest filtering with MediaTailor, MediaPackage, and CDN](#).
- **Load balancing:** Distribute load across multiple origins or ad servers using dynamic routing. For implementation guidance, see [Implement multi-Region resilience for MediaTailor with MQAR](#) and [the section called “Plan CDN integration”](#).

Supported fields

You can use dynamic variables in the following configuration fields:

- VideoContentSourceUrl
- AdDecisionServerUrl
- LivePreroll.AdDecisionServerUrl
- AdSegmentUrlPrefix
- ContentSegmentUrlPrefix
- TranscodeProfileName
- SlateAdUrl
- StartUrl
- EndUrl

The following sections describe how to use configuration aliases.

Topics

- [Creating and using configuration aliases with MediaTailor](#)
- [Configuration alias with MediaTailor use example](#)

Creating and using configuration aliases with MediaTailor

Before you start to use domain variables, you create configuration aliases for your configuration. You use the configuration aliases as domain replacement variables at session initialization time.

Restrictions

Note the following restrictions when using configuration aliases:

- All dynamic variables used in the domain must be defined as a ConfigurationAliases dynamic variable.
- The player parameter variables must be prefixed with player_params.. For example, player_params.origin_domain.
- The list of aliased values must be exhaustive for domain variables in critical URLs (VideoContentSourceUrl, AdSegmentUrlPrefix, ContentSegmentUrlPrefix).

- If a request is made for a domain variable in critical URLs that either doesn't specify the dynamic variable or uses an invalid alias, the request will fail with an HTTP 400 status code. Non-critical fields (`SlateAdUrl`, `TranscodeProfileName`, bumper URLs) will log warnings but not fail the request.

Fallback behavior for missing aliases

When configuration aliases are not found or are invalid, MediaTailor implements the following fallback behavior:

- **Domain variables:** If a domain variable alias is missing or invalid, the request fails with HTTP 400 status code. All domain variables must have valid aliases defined.
- **Non-domain variables:** For variables used in non-domain portions of URLs (such as path elements or query parameters), missing aliases result in empty string replacement.
- **Configuration validation:** MediaTailor validates that all required aliases are present during configuration creation and update operations.

Step 1: Create configuration aliases

To create configuration aliases to use for domain replacement using the MediaTailor console, perform the following procedure.

Console

To create configuration aliases using the console

1. Open the MediaTailor console at <https://console.aws.amazon.com/mediatailor/>.
2. On the **Configuration aliases** section on the **Configurations** page, choose **Add player parameter**.
3. For **Player parameter**, enter the name of the player parameter that you want to use as a dynamic variable. For example, `player_params.origin_domain`.
4. For **Aliases**, enter the aliases and their values that you want to use for the player parameter.
5. Choose **OK**.

AWS Elemental MediaTailor displays the new parameter in the table in the **Configuration aliases** section.

6. Repeat the preceding steps to add more player parameters.
7. Choose **Save**.

API

To create configuration aliases using the API

When you create or update a MediaTailor configuration, use the `ConfigurationAliases` parameter with the following JSON structure:

```
{
    "ConfigurationAliases": {
        "player_params.origin_domain": {
            "pdx": "abc.mediapackage.us-west-2.amazonaws.com",
            "iad": "xyz.mediapackage.us-east-1.amazonaws.com"
        },
        "player_params.ad_type": {
            "customized": "abc12345",
            "default": "defaultAdType"
        }
    }
}
```

Step 2: Use configuration aliases in session initialization

After you set up the configuration aliases, you can use them as replacement variables for domains in your session initialization request. This enables you to dynamically configure the domains for your session.

Example Basic configuration aliases example

Here's a basic example of a configuration that includes configuration aliases and dynamic domain variables:

```
PUT /playbackConfiguration
{
    "Name": "aliasedConfig",
    "AdDecisionServerUrl": "https://abc.execute-api.us-west-2.amazonaws.com/ads?
sid=[session.id]&ad_type=[player_params.ad_type]",
    "VideoContentSourceUrl": "https://[player_params.origin_domain].mediapackage.
[player_params.region].amazonaws.com/out/v1/[player_params.endpoint_id]",
```

```
"ConfigurationAliases": {
  "player_params.origin_domain": {
    "pdx": "abc",
    "iad": "xyz"
  },
  "player_params.region": {
    "pdx": "us-west-2",
    "iad": "us-east-1"
  },
  "player_params.endpoint_id": {
    "pdx": "abcd",
    "iad": "wxyz"
  },
  "player_params.ad_type": {
    "customized": "abc12345",
    "default": "defaultAdType"
  }
}
```

Example Session initialization with aliases

Using the preceding configuration, a session initialization request using the player variables and aliases would look similar to the following:

```
POST index.m3u8
{
  "playerParams": {
    "origin_domain": "pdx",
    "region": "pdx",
    "endpoint_id": "pdx",
    "ad_type": "customized"
  }
}
```

MediaTailor replaces the alias strings with the mapped values in the configuration aliases configuration.

The request to the ADS will look like the following:

```
https://abc.execute-api.us-west-2.amazonaws.com/ads?sid=[session.id]&ad_type=abc12345
```


The request to the origin for the manifests will look like the following:

```
https://abc.mediapackage.us-west-2.amazonaws.com/out/v1/abcd
```

Configuration alias with MediaTailor use example

The following examples show how a complete MediaTailor configuration with configuration aliases, a session initialization request with aliases, and the processing flow for aliases.

Example Complete configuration with aliases

The following example shows a complete configuration that includes configuration aliases and dynamic domain variables:

```
PUT /playbackConfiguration
{
  "Name": "aliasedConfig",
  "AdDecisionServerUrl": "https://abc.execute-api.us-west-2.amazonaws.com/ads?
sid=[session.id]&ad_type=[player_params.ad_type]",
  "VideoContentSourceUrl": "https://[player_params.origin_domain].mediapackage.
[player_params.region].amazonaws.com/out/v1/[player_params.endpoint_id]",

  "AdSegmentUrlPrefix": "https://[player_params.ad_cdn_domain]/ads/",
  "ContentSegmentUrlPrefix": "https://[player_params.content_cdn_domain]/content/",
  "TranscodeProfileName": "[player_params.transcode_profile]",
  "SlateAdUrl": "https://[player_params.slate_domain]/slate/
[player_params.slate_type].mp4",
  "StartUrl": "https://[player_params.tracking_domain]/start?session=[session.id]",
  "EndUrl": "https://[player_params.tracking_domain]/end?session=[session.id]",

  "ConfigurationAliases": {
    "player_params.origin_domain": {
      "pdx": "abc",
      "iad": "xyz"
    },
    "player_params.region": {
      "pdx": "us-west-2",
      "iad": "us-east-1"
    },
    "player_params.endpoint_id": {
      "pdx": "abcd",
      "iad": "wxyz"
    },
  },
}
```

```

    "player_params.ad_type": {
      "customized": "abc12345",
      "default": "defaultAdType"
    },
    "player_params.ad_cdn_domain": {
      "pdx": "ads-west.cdn.example.com",
      "iad": "ads-east.cdn.example.com"
    },
    "player_params.content_cdn_domain": {
      "pdx": "content-west.cdn.example.com",
      "iad": "content-east.cdn.example.com"
    },
    "player_params.transcode_profile": {
      "mobile": "mobile_optimized",
      "desktop": "high_quality",
      "tv": "4k_profile"
    },
    "player_params.slate_domain": {
      "pdx": "slate-west.example.com",
      "iad": "slate-east.example.com"
    },
    "player_params.slate_type": {
      "standard": "default_slate",
      "branded": "brand_slate"
    },
    "player_params.tracking_domain": {
      "pdx": "tracking-west.example.com",
      "iad": "tracking-east.example.com"
    }
  }
}

```

Example Session initialization with aliases

The following example shows a session initialization request that specifies the player variables and aliases:

```

POST master.m3u8
{
  "playerParams": {
    "origin_domain": "pdx",
    "region": "pdx",
    "endpoint_id": "pdx",

```

```
    "ad_type": "customized",
    "ad_cdn_domain": "pdx",
    "content_cdn_domain": "pdx",
    "transcode_profile": "mobile",
    "slate_domain": "pdx",
    "slate_type": "branded",
    "tracking_domain": "pdx"
  }
}
```

Example Parameter processing flow

In the following example, MediaTailor replaces the alias strings with the mapped values in the configuration aliases. The processing results in the following requests:

- ADS request:

```
https://abc.execute-api.us-west-2.amazonaws.com/ads?sid=[session.id]&ad_type=abc12345
```

- VideoContentSource request:

```
https://abc.mediapackage.us-west-2.amazonaws.com/out/v1/abcd
```

- AdSegmentUrlPrefix:

```
https://ads-west.cdn.example.com/ads/
```

- ContentSegmentUrlPrefix:

```
https://content-west.cdn.example.com/content/
```

- TranscodeProfileName:

```
mobile_optimized
```

- SlateAdUrl:

```
https://slate-west.example.com/slate/brand_slate.mp4
```

- StartUrl:

```
https://tracking-west.example.com/start?session=[session.id]
```

- **EndUrl:**

```
https://tracking-west.example.com/end?session=[session.id]
```

MediaTailor passing parameters to ADS

AWS Elemental MediaTailor supports setting up dynamic variables in MediaTailor requests to the ADS using the following steps.

- For information about supported formatting for query parameters, see [MediaTailor parameter reference and limitations](#).
- For configuration aliases and domain variables, see [MediaTailor configuration aliases overview](#).
- For additional customizations to the ADS request, see [Advanced usage](#).

Session initialization methods

MediaTailor supports multiple methods for session initialization and parameter passing:

1. POST with Request Body:

```
POST <master>.m3u8
{
  "adsParams": {"param1": "value1", "param2": "value2"},
  "playerParams": {"param3": "value3"}
}
```

2. Query Parameters in URL:

```
GET <master>.m3u8?ads.param1=value1&ads.param2=value2&playerParams.param3=value3
```

Important

You can only specify parameters once, at initialization time. Configuration aliases resolve to actual values before forwarding.

To pass session and player information to the ADS

1. Work with the ADS to determine the information that it needs to respond to an ad query from AWS Elemental MediaTailor.
2. Create a configuration in MediaTailor that uses a template ADS request URL that satisfies the ADS requirements. In the URL, include static parameters and include placeholders for dynamic parameters. Enter your template URL in the configuration's **Ad decision server** field.

In the following example template URL, `correlation` provides session data, and `deviceType` provides player data:

```
https://my.ads.server.com/path?  
correlation=[session.id]&deviceType=[player_params.deviceType]
```

3. On the player, configure the session initiation request for AWS Elemental MediaTailor to provide parameters for the player data. Include your parameters in the session initiation request, and omit them from subsequent requests for the session.

The type of call that the player makes to initialize the session determines whether the player (client) or MediaTailor (server) provides ad-tracking reporting for the session. For information about these two options, see [Reporting ad tracking data](#).

Make one of the following types of calls, depending on whether you want server- or client-side ad-tracking reporting. In both of the example calls, `userID` is intended for the ADS and `auth_token` is intended for the origin:

- (Option) Call for server-side ad-tracking reporting – Prefix the parameters that you want MediaTailor to send to the ADS with `ads`. Leave the prefix off for parameters that you want MediaTailor to send to the origin server:

The following examples show incoming requests for HLS and DASH to AWS Elemental MediaTailor. MediaTailor uses the `deviceType` in its request to the ADS and the `auth_token` in its request to the origin server.

HLS example:

```
GET master.m3u8?ads.deviceType=ipad&auth_token=kjhdsaf7gh
```

DASH example:

```
GET manifest.mpd?ads.deviceType=ipad&auth_token=kjhdsaf7gh
```

- (Option) Call for client-side ad-tracking reporting – Provide parameters for the ADS inside an adsParams object.

HLS example:

```
POST master.m3u8
{
  "adsParams": {
    "deviceType": "ipad"
  }
}
```

DASH example:

```
POST manifest.mpd
{
  "adsParams": {
    "deviceType": "ipad"
  }
}
```

When the player initiates a session, AWS Elemental MediaTailor replaces the variables in the template ADS request URL with the session data and the player's ads parameters. It passes the remaining parameters from the player to the origin server.

Example MediaTailor requests with ad variables

The following examples show the calls to the ADS and origin server from AWS Elemental MediaTailor that correspond to the preceding player's session initialization call examples:

- MediaTailor calls the ADS with session data and the player's device type:

```
https://my.ads.server.com/path?correlation=896976764&deviceType=ipad
```

- MediaTailor calls the origin server with the player's authorization token.
 - HLS example:

```
https://my.origin.server.com/master.m3u8?auth_token=kjhdsaf7gh
```

- DASH example:

```
https://my.origin.server.com/manifest.mpd?auth_token=kjhdsaf7gh
```

Advanced usage

You can customize the ADS request in many ways with player and session data. You only need to include the ADS host name.

The following examples show some of the ways that you can customize your request:

- Concatenate player parameters and session parameters to create new parameters. Example:

```
https://my.ads.com?key1=[player_params.value1][session.id]
```

- Use a player parameter as part of a path element. Example:

```
https://my.ads.com/[player_params.path]?key=value
```

- Use player parameters to pass both path elements and the keys themselves, rather than just values. Example:

```
https://my.ads.com/[player_params.path]?[player_params.key1]=[player_params.value1]
```

MediaTailor parameter routing for ADS and origins

AWS Elemental MediaTailor routes query parameters to different destinations based on their prefix and purpose. Understanding parameter routing is essential for implementing origin-specific functionality like time-shifted viewing with MediaPackage.

Parameter routing rules

MediaTailor uses the following routing rules for query parameters:

- **Origin parameters (no prefix):** Parameters without a specific prefix are passed through to the origin server for origin-specific functionality

- **ADS parameters (ads . prefix):** Parameters prefixed with ads . are sent to the Ad Decision Server
- **Manifest parameters (manifest . prefix):** Parameters prefixed with manifest . are used for CDN routing and authorization

Example Parameter routing example

The following session initialization demonstrates parameter routing:

```
POST /v1/session/123456789/originId/index.m3u8
{
  "adsParams": {
    "param1": "value1",
    "param2": "value2"
  },
  "manifestParams": {
    "auth_token": "abc123"
  }
}
```

In this example:

- param1 and param2 are sent to the ADS
- auth_token is used for CDN routing and authorization
- Parameters without prefixes would be passed to the origin server

Origin server parameter behavior

Parameters passed to origin servers enable origin-specific functionality such as time-shifted viewing, content filtering, and authentication.

Common origin parameter use cases

Origin parameters are commonly used for:

- **Time-shifted viewing:** start and end parameters for MediaPackage time-shifted content
- **Content authentication:** Authentication tokens required by the origin server
- **Content filtering:** Parameters that control which content variants are returned

- **Origin-specific features:** Any parameters that the origin server uses for content processing

Important

Parameters are processed at session initialization and maintained throughout the session. To modify parameters like time-shift windows, you must create a new session with updated values.

MediaTailor and MediaPackage time-shifted viewing integration

AWS Elemental MediaTailor can pass time-shifted viewing parameters to MediaPackage origins to enable startover and catch-up viewing functionality. This integration allows viewers to start watching live content from earlier points in time.

MediaPackage time-shifted viewing parameters

MediaPackage supports the following time-shifted viewing parameters that can be passed through MediaTailor:

- **start:** Epoch or ISO 8601 timestamp defining the beginning of the time-shifted manifest
- **end:** Epoch or ISO 8601 timestamp defining the end of the time-shifted manifest
- **time_delay:** Delay content availability by specified seconds
- **manifest_window_seconds:** Request a manifest shorter than the configured window

Example MediaTailor session initialization with MediaPackage time-shifted parameters

The following example shows how to initialize a session with time-shifted viewing parameters:

```
GET /v1/master/123456789/originId/index.m3u8?  
start=2024-08-26T10:00:00Z&end=2024-08-26T11:00:00Z
```

Or using explicit session initialization:

```
POST /v1/session/123456789/originId/index.m3u8  
{  
  "adsParams": {  
    "param1": "value1"
```

```
}  
}
```

With additional query parameters:

```
?start=2024-08-26T10:00:00Z&end=2024-08-26T11:00:00Z
```

Parameter behavior during sessions

Time-shifted viewing parameters have specific behavior characteristics:

- **Session initialization:** Parameters are processed when the session is created
- **Parameter persistence:** Parameters remain associated with the session throughout playback
- **Immutable after initialization:** Parameters cannot be changed during an active session
- **New session required:** To modify time-shift windows, create a new session with updated parameter values

MediaPackage startover window requirements

For time-shifted viewing to work with MediaPackage, ensure the following:

1. Configure a startover window on your MediaPackage endpoint (up to 24 hours)
2. Ensure your CDN forwards the necessary query parameters to MediaPackage
3. Use consistent playback windows across player sessions for better CDN caching
4. Verify that start and end times fall within the configured startover window

Important

When using time-shifted viewing, use consistent playback windows across player sessions rather than generating unique start or end times for each viewer. This yields better caching at the CDN and avoids potential throttling.

For complete information about MediaPackage time-shifted viewing configuration and parameters, see [Time-shifted viewing with AWS Elemental MediaPackage](#) in the *AWS Elemental MediaPackage User Guide*.

MediaTailor parameter session behavior and persistence

AWS Elemental MediaTailor processes parameters at session initialization and maintains them throughout the session lifecycle. Understanding session behavior is crucial for implementing dynamic parameter scenarios.

Session initialization methods

MediaTailor supports multiple methods for session initialization with parameters:

1. Implicit session initialization: Parameters included in the initial manifest request

```
GET /v1/master/123456789/originId/index.m3u8?
manifest.auth_token=abc123&start=2024-08-26T10:00:00Z
```

2. Explicit session initialization (POST): Parameters provided in the request body

```
POST /v1/session/123456789/originId/index.m3u8
{
  "adsParams": {"param1": "value1"},
  "manifestParams": {"auth_token": "abc123"}
}
```

3. Explicit session initialization (GET): Parameters provided as query parameters

```
GET /v1/session/123456789/originId/index.m3u8?
ads.param1=value1&manifestParams.auth_token=abc123
```

Parameter persistence and immutability

MediaTailor parameter behavior follows these rules:

- **One-time specification:** Parameters can only be specified once, at session initialization
- **Session-wide persistence:** Parameters are maintained throughout the entire session
- **Immutable after initialization:** Parameters cannot be modified after the session is created
- **Configuration alias resolution:** Aliases are resolved to actual values before forwarding to destinations

Parameter modification scenarios

To modify parameters during playback:

- **Create new session:** Initialize a new session with updated parameter values
- **Player transition:** Seamlessly transition the player to the new session
- **Parameter inheritance:** Carry forward unchanged parameters to maintain consistency

Example Modifying time-shift parameters

To change from a 1-hour window to a 2-hour window:

1. Current session: `start=2024-08-26T10:00:00Z&end=2024-08-26T11:00:00Z`
2. Create new session: `start=2024-08-26T10:00:00Z&end=2024-08-26T12:00:00Z`
3. Transition player to new session URL

Important

Multiple multivariant playlist requests for a single session do not update parameters after the first request. Parameters remain immutable for the session duration.

MediaTailor parameter reference and limitations

Before configuring dynamic ad variables, understand the parameter formatting requirements and limitations that apply to all MediaTailor configurations.

AWS Elemental MediaTailor provides comprehensive information about parameter character restrictions, length limitations, and supported formats for both manifest query parameters and ADS parameters.

Manifest query parameter character restrictions

Manifest query parameters support specific characters and have defined length limitations.

Supported characters (without URL-encoding)

You can use the following characters directly in manifest query parameters:

- Alphanumeric characters (A-Z, a-z, 0-9)

- Periods (.)
- Hyphens (-)
- Underscores (_)
- Backslashes (\)

Supported characters with URL-encoding

The following special characters are supported when URL-encoded:

- period (.) = %2E
- dash (-) = %2D
- underscore (_) = %5F
- percent (%) = %25
- tilde (~) = %7E
- forward slash (/) = %2F
- asterisk (*) = %2A
- equals (=) = %3D
- question (?) = %3F

URL-encoding support

MediaTailor supports the percent (%) sign when you use it in URL-encoding (for example, hello %20world = hello world). You can use any URL-encoded characters, as long as they are valid URL-encodings according to the HTTP specification.

Unsupported characters

You cannot use the following characters in manifest query parameters without URL-encoding: :, ?, &, =, %, / (forward slash).

Important

MediaTailor doesn't support double characters such as %%% or ==. You can't use full URLs as manifest query parameter values due to character restrictions.

Length limitations

The total length of all manifest query parameters (keys and values combined) must not exceed 2000 characters.

ADS parameter length limitations

The following length limitations apply to parameters used in requests to the ADS:

- **ADS parameter name:** Maximum 10,000 characters
- **ADS parameter value:** Maximum 25,000 characters
- **ADS URL:** Maximum 25,000 characters

MediaTailor parameter troubleshooting guide

AWS Elemental MediaTailor provides guidance for troubleshooting common parameter-related issues in MediaTailor, including character restrictions, URL encoding problems, and configuration alias errors.

Character restriction errors

Parameter values that contain unsupported characters may cause errors or unexpected behavior.

Common symptoms

The following symptoms may indicate character restriction issues:

- Parameters not appearing in manifest URLs
- HTTP 400 errors during session initialization
- Truncated or corrupted parameter values
- ADS requests failing due to malformed URLs

Resolution steps

To resolve character restriction errors:

1. Review parameter values for unsupported characters: :, ?, &, =, %, /

2. Apply proper URL-encoding for special characters (see [MediaTailor parameter reference and limitations](#))
3. Avoid double characters such as %%% or ==
4. Consider alternative parameter formats if full URLs cannot be used

Example URL encoding example

Instead of using:

```
manifest.redirect_url=https://example.com/path?param=value
```

Use URL-encoded format:

```
manifest.redirect_url=https%3A%2F%2Fexample.com%2Fpath%3Fparam%3Dvalue
```

Length limitation errors

Parameters that exceed length limits may be truncated or cause errors.

Length limits

The following length limits apply (see [MediaTailor parameter reference and limitations](#) for complete details):

- Manifest query parameters (total): 2000 characters
- ADS parameter names: 10,000 characters
- ADS parameter values: 25,000 characters
- ADS URLs: 25,000 characters

Resolution strategies

To handle length limitations:

1. Use shorter parameter names and values where possible
2. Split large parameter values into multiple smaller parameters
3. Use configuration aliases to map short aliases to longer values (see [MediaTailor configuration aliases overview](#))

4. Consider using external storage for large data with parameter references

Configuration alias errors

Configuration alias issues can result in HTTP 400 errors or unexpected parameter values.

Common configuration alias errors

The following errors commonly occur with configuration aliases:

- HTTP 400 error: Missing or invalid alias value
- Domain variables not resolving correctly
- Player parameters not being replaced with alias values

Resolution checklist

To resolve configuration alias errors:

1. Verify all domain variables are defined as `ConfigurationAliases`
2. Ensure player parameter variables use `player_params.` prefix
3. Confirm the list of aliased values is exhaustive for domain variables in critical URLs (`VideoContentSourceUrl`, `AdSegmentUrlPrefix`, `ContentSegmentUrlPrefix`)
4. Check that session initialization requests specify valid alias values
5. Validate JSON structure of `ConfigurationAliases` parameter

For detailed troubleshooting guidance, see [MediaTailor configuration aliases troubleshooting guide](#).

Example Configuration alias validation

Ensure your configuration includes all required aliases:

```
"ConfigurationAliases": {
  "player_params.origin_domain": {
    "pdx": "abc.mediapackage.us-west-2.amazonaws.com",
    "iad": "xyz.mediapackage.us-east-1.amazonaws.com"
    // Must include all possible values used in session initialization
  }
}
```



```
}
```

Parameter processing flow issues

Understanding the parameter processing flow helps troubleshoot issues with parameter forwarding and transformation.

Parameter processing order

MediaTailor processes parameters in the following order:

1. Session initialization parameter validation
2. Configuration alias resolution (if applicable)
3. Parameter filtering (ADS vs origin vs manifest)
4. URL encoding and formatting
5. Parameter application to URLs

Debugging parameter flow

To debug parameter processing issues:

1. Verify parameters are correctly specified in session initialization
2. Check that configuration aliases resolve to expected values
3. Confirm parameters appear in the correct URLs (manifest, ADS, origin)
4. Validate URL encoding is applied correctly

Example Parameter flow example

Session initialization:

```
POST master.m3u8
{
  "playerParams": {"origin_domain": "pdx"},
  "manifestParams": {"test": "123"}
}
```

After alias resolution and processing:

- Origin request: `https://abc.mediapackage.us-west-2.amazonaws.com/out/v1/abcd`
- Manifest URL: `/v1/master/.../index.m3u8?aws.sessionId=session&test=123`

Security considerations and best practices

MediaTailor implements security measures for parameter handling to prevent common security issues.

Security measures

MediaTailor implements the following security measures:

1. Input size limitations to prevent database bloat
2. Proper encoding and sanitization of user input
3. URL-encoding of input to prevent response corruption

Best practices

Follow these best practices for secure parameter handling:

- Validate parameter values on the client side before sending
- Use configuration aliases to limit possible parameter values
- Avoid including sensitive information in parameters
- Monitor parameter usage for unusual patterns
- Keep parameter values within recommended length limits

MediaTailor configuration aliases troubleshooting guide

AWS Elemental MediaTailor provides systematic troubleshooting guidance for common configuration aliases issues and error scenarios.

Configuration alias validation errors

When configuration aliases are missing or invalid, MediaTailor returns specific error responses to help identify the issue.

Common error scenarios

The following table describes common configuration alias errors and their resolution steps:

Error	Cause	Resolution
HTTP 400: Invalid player parameter alias	Player parameter value not found in ConfigurationAliases	Verify that the player parameter value exists as a key in the corresponding ConfigurationAliases mapping
HTTP 400: Missing required configuration alias	Domain variable used without corresponding ConfigurationAliases entry	Add the missing player parameter to ConfigurationAliases with all required alias mappings
HTTP 400: Configuration validation failed	ConfigurationAliases structure is malformed or incomplete	Validate JSON structure and ensure all domain variables have corresponding aliases
Empty string replacement in URLs	Non-domain variable alias not found	Add missing alias mapping or provide default value in ConfigurationAliases

Validation checklist

Use the following checklist to validate your configuration aliases setup:

1. **Domain variable coverage:** Ensure all variables used in domain portions of URLs have corresponding ConfigurationAliases entries
2. **Alias completeness:** Verify that all possible player parameter values are included as keys in the alias mappings
3. **JSON structure:** Validate that the ConfigurationAliases JSON is properly formatted and nested
4. **Parameter naming:** Confirm that all player parameters use the `player_params.` prefix
5. **Value consistency:** Ensure alias values are valid for their intended use (URLs, profile names, etc.)

Debugging configuration alias resolution

Follow this systematic approach to debug configuration alias resolution issues.

Step-by-step debugging methodology

Use the following steps to identify and resolve configuration alias issues:

Configuration alias debugging procedure

1. **Verify configuration structure:** Confirm that your playback configuration includes properly formatted ConfigurationAliases

```
{
  "ConfigurationAliases": {
    "player_params.example_param": {
      "alias1": "value1",
      "alias2": "value2"
    }
  }
}
```

2. **Check player parameter format:** Ensure session initialization includes correctly formatted player parameters

```
{
  "playerParams": {
    "example_param": "alias1"
  }
}
```

3. **Validate alias mapping:** Confirm that the player parameter value ("alias1") exists as a key in the ConfigurationAliases mapping
4. **Test with simple configuration:** Start with a minimal configuration to isolate the issue
5. **Monitor error responses:** Check MediaTailor error responses for specific validation messages
6. **Verify resolved URLs:** Confirm that the final resolved URLs are valid and accessible

Configuration aliases best practices

Follow these best practices to ensure reliable configuration alias implementation.

Security considerations

Implement the following security measures when you use configuration aliases:

- **Input validation:** Validate all player parameter values before using them in alias resolution

- **Alias value sanitization:** Ensure alias values contain only expected characters and formats
- **Domain restrictions:** Limit domain aliases to trusted, controlled domains
- **Access control:** Restrict configuration modification to authorized personnel only

Performance optimization

Optimize configuration alias performance with these recommendations:

- **Minimize alias count:** Use only necessary aliases to reduce processing overhead
- **Efficient naming:** Use clear, consistent naming conventions for aliases and parameters
- **Default values:** Provide sensible default aliases for common use cases
- **Configuration caching:** Leverage MediaTailor's configuration caching for improved performance

Maintenance and monitoring

Maintain reliable configuration alias operations with these practices:

- **Regular validation:** Periodically validate that all alias mappings are current and functional
- **Error monitoring:** Monitor for HTTP 400 errors related to missing or invalid aliases
- **Documentation:** Maintain clear documentation of all alias mappings and their purposes
- **Testing procedures:** Implement comprehensive testing for all alias combinations

MediaTailor manifest query parameters

AWS Elemental MediaTailor handles query parameters for different purposes: manifest query parameters for CDN routing and authorization, and other query parameters that may be used for origin-specific functionality.

AWS Elemental MediaTailor preserves query parameters from session initialization and appends them to personalized manifest URLs and other assets. Use this functionality when you have a Content Delivery Network (CDN) between MediaTailor and the client player.

Use manifest query parameters when your CDN needs the query parameters for the following:

- Dynamic routing to different MediaTailor endpoints
- Token authorization

Client-side vs CDN behavior

MediaTailor appends query parameters for client-side reporting endpoints, but does not append them for CDN segments. The updated functionality provides more comprehensive support for query parameters across various MediaTailor assets, enhancing flexibility for CDN routing and authorization use cases.

MediaTailor appends query parameters for client-side reporting endpoints, but it doesn't append the query parameters for the CloudFront (or other CDN) segments.

To use parameter preservation, contact [AWS Support](#) to request that manifest query parameter pass through be enabled.

The behavior varies between HLS and DASH, and between explicit and implicit session initialization. The following topics describe how to configure session initialization requests so that MediaTailor passes through parameters to the manifest:

MediaTailor query parameter handling for origins

AWS Elemental MediaTailor handles query parameters differently depending on their purpose. Manifest query parameters (prefixed with `manifest.`) are used for CDN routing and authorization, while other query parameters can be used for origin-specific functionality.

Time-shifted viewing with MediaPackage

For time-shifted viewing functionality with MediaPackage, you can include `start` and `end` parameters in your requests. These parameters define specific content windows for startover and catch-up viewing.

Example time-shifted viewing request

Include `start` and `end` parameters in your manifest request for time-shifted viewing:

```
GET /v1/master/111122223333/originId/index.m3u8?  
start=2024-08-26T10:00:00Z&end=2024-08-26T11:00:00Z
```

Parameter behavior during sessions

Query parameters are processed at session initialization. For time-shifted viewing or other origin-specific features:

- **Session initialization:** Include required parameters with the initial manifest request
- **Parameter persistence:** Parameters are associated with the session and maintained throughout playback
- **Changing parameters:** To modify time-shift windows or other parameters, create a new session with updated values

Important

The specific handling of query parameters depends on your origin configuration and the parameters your content origin supports. For MediaPackage integration, ensure your CDN is configured to forward the necessary query parameters as described in [Configure essential query parameters](#).

MediaTailor parameter character restrictions and URL-encoding

AWS Elemental MediaTailor supports specific characters in manifest query parameters. You can use URL-encoding for special characters.

Supported characters with URL-encoding

The following special characters are supported with URL-encoding:

- period (.) = %2E
- dash (-) = %2D
- underscore (_) = %5F
- percent (%) = %25
- tilde (~) = %7E
- forward slash (/) = %2F
- asterisk (*) = %2A
- equals (=) = %3D
- question (?) = %3F

URL-encoding support

MediaTailor supports the percent (%) sign when you use it in URL-encoding (for example, hello %20world = hello world). You can use any URL-encoded characters, as long as they are valid URL-encodings according to the HTTP specification.

Important

MediaTailor doesn't support double characters such as %%% or ==.

Security considerations

MediaTailor implements the following security measures for parameter handling:

1. Input size limitations to prevent database bloat
2. Proper encoding and sanitization of user input
3. URL-encoding of input to prevent response corruption

Topics

- [MediaTailor HLS implicit session initialization](#)
- [MediaTailor DASH implicit session initialization](#)
- [MediaTailor HLS and DASH explicit session initialization](#)
- [MediaTailor protocol-specific parameter behavior](#)
- [MediaTailor CDN integration and parameter routing](#)

MediaTailor HLS implicit session initialization

AWS Elemental MediaTailor includes query parameters in links to MediaTailor resources when your request includes query parameters with the key `manifest.*`. The following example shows this request format:

```
GET /v1/master/111122223333/originId/index.m3u8?manifest.test=123&other=456
```

Links don't include the `manifest.` prefix.

Parameter application in HLS

For HLS implicit sessions, MediaTailor applies parameters to the following locations in the manifest hierarchy:

- Multivariant playlist URLs
- Media playlist URLs
- Content segment URLs
- Ad segment URLs
- HLS initialization URLs

Example multivariant playlist

The following example shows how MediaTailor includes the query parameters in the URL for the multivariant playlist:

```
#EXTM3U
#EXT-X-VERSION:3
#EXT-X-INDEPENDENT-SEGMENTS
#EXT-X-MEDIA:LANGUAGE="eng",AUTOSELECT=YES,FORCED=NO,TYPE=SUBTITLES,URI="../../manifest/111122223333/originId/session/1.m3u8?test=123",GROUP-ID="subtitles",DEFAULT=YES,NAME="caption_1"
#EXT-X-STREAM-INF:CODECS="avc1.640029,mp4a.40.2",AVERAGE-BANDWIDTH=2525657,RESOLUTION=960x540,SUBTITLES="subtitles",FRAME-RATE=29.97,BANDWIDTH=2665212
../../manifest/111122223333/originId/session/0.m3u8?test=123
```

Example media playlist

The following example shows how MediaTailor includes the query parameters in the URLs for the content segments:

```
#EXTM3U
#EXT-X-VERSION:6
#EXT-X-TARGETDURATION:7
#EXT-X-MEDIA-SEQUENCE:28716269
#EXT-X-DISCONTINUITY-SEQUENCE:0
#EXTINF:6.006,
https://origin.com/contentSegment_1.ts?originQueryParam=foo&test=123
#EXT-X-DISCONTINUITY
#EXTINF:6.006,
../../segment/111122223333/originId/session/0/2?test=123
```

MediaTailor DASH implicit session initialization

AWS Elemental MediaTailor creates a session for the client and redirects it with query parameters when the client makes a manifest request without a session. The following example shows this request format:

```
GET /v1/dash/111122223333/originId/index.mpd?manifest.test=123&other=456
```

MediaTailor creates a session for the client and redirects it with the query parameters:

```
/v1/dash/111122223333/originId/index.mpd?sessionId=session&test=123
```

Parameter application in DASH

The DASH manifest response includes the query parameters in various locations, including content segments, ad segments, and initialization URLs. MediaTailor applies parameters to the following:

- DASH manifest Location elements
- SegmentTemplate initialization attributes
- SegmentTemplate media attributes
- Content segment URLs
- Ad segment URLs

When the client makes the request, MediaTailor responds with a DASH manifest similar to the following example. The first period is a content period, so MediaTailor doesn't insert the manifest query parameter there. In the second period, which is an ad period, MediaTailor inserts the manifest query parameter into the SegmentTemplate element's initialization attribute and media attribute. The Location element also has the manifest query parameters.

```
<?xml version="1.0" encoding="UTF-8"?>
<MPD availabilityStartTime="2018-07-27T09:48:23.634000+00:00"
  id="201" minBufferTime="PT30S" minimumUpdatePeriod="PT15S"
  profiles="urn:mpeg:dash:profile:isoff-live:2011" publishTime="2023-02-14T23:37:43"
  suggestedPresentationDelay="PT25.000S" timeShiftBufferDepth="PT56.997S" type="dynamic"
  xmlns="urn:mpeg:dash:schema:mpd:2011" xmlns:scte35="urn:scte:scte35:2013:xml"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="urn:mpeg:dash:schema:mpd:2011 http://standards.iso.org/ittf/
PubliclyAvailableStandards/MPEG-DASH_schema_files/DASH-MPD.xsd">
  <BaseURL>https://origin.com/contentSegments/</BaseURL>
```

```

<Location>https://mediatailor.com/v1/dash/111122223333/originId/index.mpd?
test=123&aws.sessionId=session</Location>
  <Period duration="PT29.963S" id="28737823" start="PT143732873.178S">
    <AdaptationSet bitstreamSwitching="true" mimeType="video/mp4"
segmentAlignment="true" startWithSAP="1" subsegmentAlignment="true"
subsegmentStartsWithSAP="1">
      <Representation bandwidth="2200000" codecs="avc1.640029"
frameRate="30000/1001" height="540" id="1" width="960">
        <SegmentTemplate initialization="index_video_7_0_init.mp4?
m=1611174111" media="index_video_7_0_${Number$.mp4?m=1611174111"
presentationTimeOffset="4311986195351" startNumber="28737828" timescale="30000">
          <SegmentTimeline>
            <S d="180180" t="4311986911066"/>
            <S d="3003" t="4311987091246"/>
          </SegmentTimeline>
        </SegmentTemplate>
      </Representation>
    </AdaptationSet>
  </Period>
  <Period id="28737829_1" start="PT39925H48M23.141S">
    <BaseURL>https://mediatailor.com/v1/
dashsegment/111122223333/originId/session/28737829/28737829_1/</BaseURL>
    <AdaptationSet bitstreamSwitching="false" frameRate="30000/1001"
mimeType="video/mp4" segmentAlignment="true" startWithSAP="1"
subsegmentAlignment="true" subsegmentStartsWithSAP="1">
      <SegmentTemplate startNumber="1" timescale="90000"/>
      <Representation bandwidth="2200000" codecs="avc1.64001f" height="540"
id="1" width="960">
        <SegmentTemplate initialization="asset_540_2_0init.mp4?test=123"
media="asset_540_2_0_${Number%09d$.mp4?test=123" startNumber="1" timescale="90000">
          <SegmentTimeline>
            <S d="180180" r="6" t="0"/>
            <S d="87087" t="1261260"/>
          </SegmentTimeline>
        </SegmentTemplate>
      </Representation>
    </AdaptationSet>
  </Period>
</MPD>

```

MediaTailor HLS and DASH explicit session initialization

AWS Elemental MediaTailor includes the `manifestParams` as query parameters in the multivariant playlist and tracking URLs in the response when the client makes an explicit session initialization request.

Session initialization methods

For explicit session initialization, you can use either POST with request body or GET with query parameters:

1. POST with Request Body:

```
POST /v1/session/111122223333/originId/index.m3u8
{
  "adsParams": {"param1": "value1", "param2": "value2", "param3": "value3"},
  "manifestParams": {"test": "123"}
}
```

2. GET with Query Parameters:

```
GET /v1/session/111122223333/originId/index.m3u8?
ads.param1=value1&ads.param2=value2&manifestParams.test=123
```

Example session initialization request

```
POST /v1/session/111122223333/originId/index.m3u8
{
  "adsParams": {
    "param1": "value1",
    "param2": "value2",
    "param3": "value3"
  },
  "manifestParams": {
    "test": "123"
  },
  "reportingMode": "client"
}
```

Example manifest and tracking response

```
{
  "manifestUrl": "/v1/master/111122223333/originId/index.m3u8?
aws.sessionId=session&test=123",
  "trackingUrl": "/v1/tracking/111122223333/originId/session?test=123"
}
```

Manifest responses for the session have the specific manifestParams in MediaTailor URLs similar to the previously described implicit session-initialization workflows. The key difference is that manifest parameters for explicit session initialization don't start with manifest..

Manifest query parameters are immutable and you can only set them on session initialization. If a client makes multiple multivariant playlist requests for a single session, MediaTailor doesn't update the manifest query parameters after the first request.

Parameter processing flow

You can only specify parameters once, at initialization time. Configuration aliases resolve to actual values before forwarding. For example: player_params.ad_type=customized resolves to ad_type=abc12345 based on the ConfigurationAliases configuration.

MediaTailor protocol-specific parameter behavior

AWS Elemental MediaTailor handles manifest query parameters differently for HLS and DASH protocols. Each protocol type has specific application locations and processing methods.

HLS vs DASH parameter handling comparison

The following table compares how MediaTailor handles manifest query parameters across HLS and DASH protocols:

Aspect	HLS Behavior	DASH Behavior
Parameter application	Applied directly to manifest URLs and segment URLs	Applied to Location elements, SegmentTemplate attributes, and segment URLs
Manifest hierarchy	Multivariant playlist → Media playlists→ Segments	MPD → Periods → AdaptationSets → Representations

Aspect	HLS Behavior	DASH Behavior
Initialization URLs	Applied to HLS init URLs when present	Applied to SegmentTemplate initialization attributes
Session handling	Parameters preserved across playlist updates	Parameters included in MPD Location element for session continuity
Ad segment handling	Applied to ad segment URLs in media playlists	Applied to ad period SegmentTemplate media attributes

Parameter application locations

MediaTailor applies manifest query parameters to the following locations:

HLS parameter application

For HLS streams, MediaTailor applies manifest query parameters to:

- **Multivariant playlist URLs:** Parameters are appended to media playlist references
- **Media playlist URLs:** Parameters are included in segment URLs within media playlists
- **Content segment URLs:** All content segments include the manifest query parameters
- **Ad segment URLs:** Ad segments receive parameters for CDN routing and authorization
- **HLS initialization URLs:** Init segments include parameters when present in the stream
- **Slate segment URLs:** Slate content includes parameters for consistent CDN behavior

Example HLS parameter application example

Given the session initialization:

```
GET /v1/master/123456789/originId/index.m3u8?
manifest.auth_token=abc123&manifest.region=us-west
```

The multivariant playlist includes parameters in media playlist references:

```
#EXTM3U
```

```
#EXT-X-VERSION:3
#EXT-X-STREAM-INF:BANDWIDTH=2665212,RESOLUTION=960x540
../.../manifest/123456789/originId/session/0.m3u8?auth_token=abc123&region=us-west
```

The media playlist includes parameters in segment URLs:

```
#EXTM3U
#EXT-X-VERSION:6
#EXT-X-TARGETDURATION:7
#EXTINF:6.006,
https://origin.com/segment1.ts?auth_token=abc123&region=us-west
#EXTINF:6.006,
../.../.../segment/123456789/originId/session/0/2?auth_token=abc123&region=us-west
```

DASH parameter application

For DASH streams, MediaTailor applies manifest query parameters to:

- **MPD Location elements:** The Location element includes parameters for manifest refresh requests
- **SegmentTemplate initialization attributes:** Init segment URLs include parameters
- **SegmentTemplate media attributes:** Media segment URL templates include parameters
- **Content segment URLs:** All content segments generated from templates include parameters
- **Ad segment URLs:** Ad period segments include parameters for CDN integration
- **Server-side reporting redirects:** 302 redirects to ad segments preserve parameters

Example DASH parameter application example

Given the session initialization:

```
GET /v1/dash/123456789/originId/index.mpd?
manifest.auth_token=abc123&manifest.region=us-west
```

The DASH manifest includes parameters in multiple locations:

```
<MPD>
  <Location>https://mediatailor.com/v1/dash/123456789/originId/index.mpd?
  auth_token=abc123&region=us-west&aws.sessionId=session</Location>
```

```
<Period>
  <AdaptationSet>
    <Representation>
      <SegmentTemplate
        initialization="init.mp4?auth_token=abc123&region=us-west"
        media="segment_${Number}.mp4?auth_token=abc123&region=us-west"/>
      </Representation>
    </AdaptationSet>
  </Period>
</MPD>
```

MediaTailor CDN integration and parameter routing

AWS Elemental MediaTailor manifest query parameters enable sophisticated CDN integration scenarios. You can use them for dynamic routing, authorization, and load balancing.

CDN routing use cases

Common CDN integration scenarios that benefit from manifest query parameters include the following:

- **Geographic routing:** Route requests to region-specific MediaTailor endpoints based on viewer location
- **Token-based authorization:** Pass authorization tokens through the CDN to MediaTailor for secure content access
- **Load balancing:** Distribute traffic across multiple MediaTailor endpoints using CDN routing logic
- **A/B testing:** Route different user segments to different MediaTailor configurations for testing
- **Device-specific optimization:** Route requests based on device type or capabilities

Parameter preservation across CDN layers

MediaTailor ensures that manifest query parameters are preserved across multiple CDN layers and request types:

1. **Initial request:** Parameters are extracted from the session initialization request
2. **Manifest generation:** Parameters are applied to all relevant URLs in the manifest
3. **Segment requests:** Parameters are included in all segment URLs for consistent CDN behavior
4. **Ad insertion:** Parameters are preserved during ad insertion and segment replacement

Example CDN authorization flow

The following example demonstrates a complete CDN authorization flow using manifest query parameters:

1. Client requests manifest with authorization token:

```
GET https://cdn.example.com/mediatailor/v1/master/123456789/originId/index.m3u8?
manifest.auth_token=jwt_token_here&manifest.user_id=12345
```

2. CDN forwards request to MediaTailor with parameters:

```
GET https://mediatailor.amazonaws.com/v1/master/123456789/originId/index.m3u8?
manifest.auth_token=jwt_token_here&manifest.user_id=12345
```

3. MediaTailor generates manifest with parameters applied to all URLs:

```
#EXTM3U
#EXT-X-STREAM-INF:BANDWIDTH=2665212
../.../manifest/123456789/originId/session/0.m3u8?
auth_token=jwt_token_here&user_id=12345
```

4. Subsequent segment requests include parameters for CDN authorization:

```
GET https://cdn.example.com/mediatailor/segment/123456789/originId/session/0/1?
auth_token=jwt_token_here&user_id=12345
```

Reporting ad tracking data

MediaTailor provides two options for tracking and reporting on how much of an ad a viewer has watched. In the server-side ad reporting approach, MediaTailor tracks the ad and sends beacons (tracking signals) directly to the ad server. Alternatively, in the client-side tracking approach, the client player (the user's device) tracks the ad and sends the beacons to the ad server. The type of ad reporting used in a playback session depends on the specific request the player makes to initiate the session in MediaTailor.

For information about passing session and player data to the ad server using dynamic variables, see [MediaTailor dynamic ad variables for ADS requests](#). For details about session initialization parameters, see [MediaTailor manifest query parameters](#).

Topics

- [MediaTailor server-side ad tracking and reporting](#)
- [Client-side ad tracking](#)

MediaTailor server-side ad tracking and reporting

AWS Elemental MediaTailor defaults to server-side reporting for comprehensive ad tracking and measurement. With server-side reporting, when the player requests an ad URL from the manifest, the service reports ad consumption directly to the ad tracking URL. After the player initializes a playback session with MediaTailor, no further input is required from you or the player to perform server-side reporting. As each ad is played back, MediaTailor sends beacons to the ad server to report how much of the ad has been viewed. MediaTailor sends beacons for the start of the ad and for the ad progression in quartiles: the first quartile, midpoint, third quartile, and ad completion.

To perform server-side ad reporting

- From the player, initialize a new MediaTailor playback session using a request in one of the following formats, according to your protocol:
 - Example: HLS format

```
GET <mediatailorURL>/v1/master/<hashed-account-id>/<origin-id>/<asset-id>?  
ads.<key-value-pairs-for-ads>&<key-value-pairs-for-origin-server>
```

- Example: DASH format

```
GET <mediatailorURL>/v1/dash/<hashed-account-id>/<origin-id>/<asset-id>?ads.<key-  
value-pairs-for-ads>&<key-value-pairs-for-origin-server>
```

The key-value pairs are the dynamic targeting parameters for ad tracking. For information about adding parameters to the request, see [the section called “Dynamic ad variables”](#).

AWS Elemental MediaTailor responds to the request with the manifest URL. The manifest contains URLs for the media manifests. The media manifests contain embedded links for ad segment requests.

Note

When MediaTailor encounters a double-slash (//) in a tracking URL, it collapses the slashes to one (/).

When the player requests playback from an ad segment URL (/v1/segment path), AWS Elemental MediaTailor sends the appropriate beacon to the ad server through the ad tracking URLs. At the same time, the service issues a redirect to the actual *.ts ad segment. The ad segment is either in the Amazon CloudFront distribution where MediaTailor stores transcoded ads, or in the content delivery network (CDN) where you have cached the ad.

The following sections provide more information about working with server-side ad tracking from MediaTailor.

Topics

- [Server-side tracking with server-guided ad insertion \(SGAI\)](#)
- [Server-side tracking beacon glossary](#)
- [Server-side tracking timing and caching behavior](#)
- [MediaTailor server-side tracking features and capabilities](#)

Server-side tracking with server-guided ad insertion (SGAI)

When you use server-guided ad insertion (SGAI), server-side tracking uses a *sessionless beaconing* mechanism that differs from the stitched-mode approach described above. Instead of MediaTailor stitching ad segments into the content manifest (where it tracks /v1/segment requests), SGAI returns ad references as separate playlists in an asset list response with beacon metadata embedded in the ad URIs.

How sessionless server-side beaconing works

The following steps describe how server-side beaconing works for SGAI sessions:

1. **Session initialization:** The player requests the HLS multivariant playlist with `aws.insertionMode=GUIDED`. Server-side reporting is the default (no `aws.reportingMode` parameter is needed). Unlike stitched mode, the session initialization response does *not* include a `trackingUrl`.

2. **Cacheable manifest:** MediaTailor returns a cacheable manifest containing EXT-X-DATERANGE tags with CLASS="com.apple.hls.interstitial" and X-ASSET-LIST attributes pointing to the MediaTailor interstitial asset list endpoint.
3. **Asset list with beacon metadata:** When the player encounters an ad break, it fetches the asset list. MediaTailor returns a JSON response where each ad URI includes encrypted beacon metadata:

```
{
  "ASSETS": [
    {
      "DURATION": 30.0,
      "URI": "https://cdn.example.com/ad/master.m3u8?
awsBeaconData=<encrypted>&awsBeaconDomain=<MediaTailor-
endpoint>&awsConfigurationName=<config-name>"
    }
  ]
}
```

When server-side reporting is active, the response does *not* include a TRACKING section. The ad URIs carry all beacon data.

4. **HLS variable substitution:** The player fetches the ad multivariant playlist. The ad manifest uses #EXT-X-DEFINE:QUERYPARAM directives to pass the beacon parameters from the URI query string into segment URLs via HLS variable substitution:

```
#EXTM3U
#EXT-X-DEFINE:QUERYPARAM="awsBeaconData"
#EXT-X-DEFINE:QUERYPARAM="awsBeaconDomain"
#EXT-X-DEFINE:QUERYPARAM="awsConfigurationName"
#EXTINF:5.0,
{$awsBeaconDomain}/segment/hash/{$awsConfigurationName}/{$awsBeaconData}/0/0?
aws.segmentRelativePath=asset_00001.ts
```

The player resolves the {\$awsBeaconData}, {\$awsBeaconDomain}, and {\$awsConfigurationName} variables using the values from the ad manifest URI query string, and then requests each ad segment through MediaTailor.

5. **Beacon firing on segment request:** As the player requests each ad segment, the request routes through MediaTailor. The service decrypts the beacon data, determines the segment's position within the ad (impression, first quartile, midpoint, third quartile, or complete), and fires the

appropriate VAST tracking beacon to the ad server. MediaTailor then redirects the player to the actual ad content segment.

Player requirements for SGAI server-side beaconing

To use server-side beaconing with SGAI, your player must meet the following requirements:

- HLS version 11 or later
- Support for EXT-X-DATERANGE with CLASS attribute for HLS Interstitials
- Support for #EXT-X-DEFINE:QUERYPARAM variable substitution (RFC 8216bis). The player must percent-decode the query parameter values before substituting them into segment URLs.

Note

SGAI server-side beaconing is currently supported for HLS only. DASH is not yet supported for SGAI server-side beaconing.

Comparison with stitched-mode server-side tracking

The following table summarizes how server-side tracking differs between stitched and server-guided ad insertion:

Aspect	Stitched (SSAI)	Server-guided (SGAI)
Manifest cacheability	Per-session, not cacheable	Cacheable, shared across viewers
Ad segment routing	Through /v1/segment/ using the session ID	Through /v1/segment/ using an encrypted beacon data blob
Session state for beacons	Stored per session in MediaTailor	Sessionless — all state is carried in the encrypted awsBeaconData parameter
Tracking URL at session init	Returned in the session initialization response	Not provided — beacon data is embedded in ad URIs in each asset list response

Aspect	Stitched (SSAI)	Server-guided (SGAI)
DASH support	Supported	Not yet supported

Note

For live SGA sessions, you can enable manifest-based ad prefetching using `aws.guidedPrefetchMode=MANIFEST`. This is separate from the schedule-based prefetch API used with stitched (SSAI) sessions. For details, see [Guided prefetch with manifest heartbeating](#).

Server-side tracking beacon glossary

MediaTailor server-side tracking uses a standardized set of beacons to report ad viewing progress to ad servers and verification services. These beacons align with Interactive Advertising Bureau (IAB) standards for video ad measurement and provide accurate reporting of ad impressions and completion rates.

Server-side tracking beacon types

Beacon Type	When Fired	Purpose	Timing Details
Impression	When the player requests the first ad segment	Indicates that ad content has begun loading and is about to be displayed to the viewer	Fired on the first <code>/v1/segment</code> request for an ad. Aligns with IAB guidelines requiring ad content to begin loading before counting an impression. See Server-side tracking workflow for the complete sequence.
Start	When the player begins rendering the ad content	Confirms that ad playback has actually started	Typically fired simultaneously with the impression beacon on the first segment request, but represents the actual start of ad rendering. This distinction is important for

Beacon Type	When Fired	Purpose	Timing Details
			verification services that track both impression and start events separately.
First Quartile	When the player reaches 25% of the ad duration	Measures continued ad viewing through the first quarter of the ad	Fired when the player requests the segment that contains the 25% point of the ad duration. For example, in a 20-second ad with 2-second segments, this would typically fire on the request for the 3rd segment (at approximately 4-6 seconds into the ad).
Midpoint	When the player reaches 50% of the ad duration	Measures continued ad viewing through half of the ad	Fired when the player requests the segment that contains the 50% point of the ad duration. For example, in a 20-second ad with 2-second segments, this would typically fire on the request for the 5th segment (at approximately 8-10 seconds into the ad).
Third Quartile	When the player reaches 75% of the ad duration	Measures continued ad viewing through three-quarters of the ad	Fired when the player requests the segment that contains the 75% point of the ad duration. For example, in a 20-second ad with 2-second segments, this would typically fire on the request for the 8th segment (at approximately 14-16 seconds into the ad).

Beacon Type	When Fired	Purpose	Timing Details
Complete	When the player reaches the end of the ad	Confirms that the entire ad was delivered to the viewer	Fired when the player requests the final segment of the ad. This indicates that the viewer has potentially seen the entire ad content. For example, in a 20-second ad with 2-second segments, this would typically fire on the request for the 10th segment (at approximately 18-20 seconds into the ad).

Note

The exact timing of beacon firing depends on the segment duration and ad length. MediaTailor calculates the appropriate segment request that corresponds to each quartile position based on the specific ad duration and segment structure.

Server-side tracking timing and caching behavior

In server-side reporting, MediaTailor fires tracking events based on actual segment requests from the player, not on manifest parsing or pre-loading activities. This approach ensures accurate impression counting that aligns with industry standards for video ad measurement.

Key timing principles

MediaTailor server-side tracking follows these fundamental timing principles:

- **Tracking events fire on actual segment requests** - Beacons are sent only when the player makes HTTP requests to `/v1/segment` URLs, not during manifest parsing or caching.
- **Player caching and pre-loading of manifests does NOT trigger events** - Players can parse, cache, or pre-load manifest information without generating any tracking events.

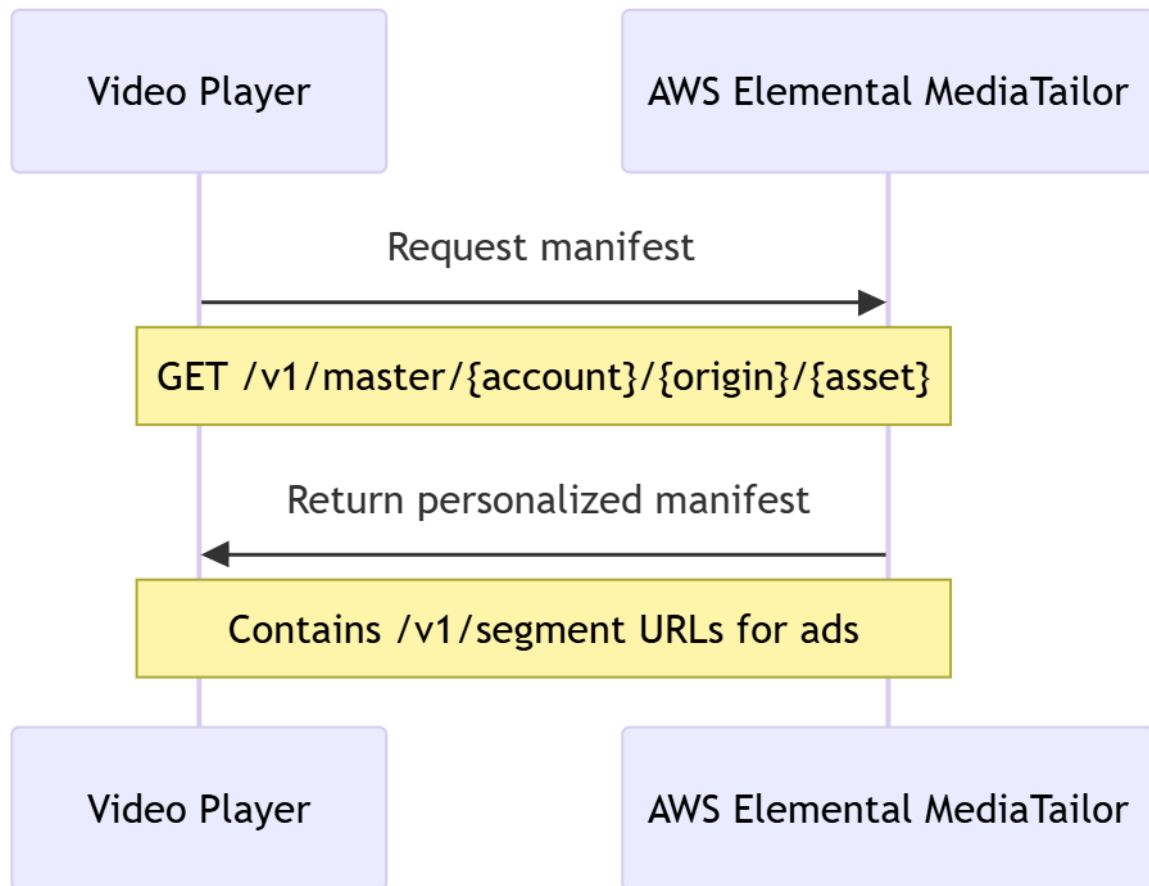
- **Segment pre-fetching *will* trigger events** - If players pre-fetch actual ad segments before playback, this follows industry standard behavior where segment requests constitute valid impressions.
- **Each `/v1/segment` request triggers appropriate beacon** - The specific tracking event (impression, quartile, completion) is determined by the ad position and segment being requested.
- **Timing aligns with IAB standards** - The approach follows Interactive Advertising Bureau guidelines for video ad measurement and impression counting.

Server-side tracking workflow

The following diagrams illustrate the complete server-side tracking workflow, showing when tracking events are fired in relation to player requests:

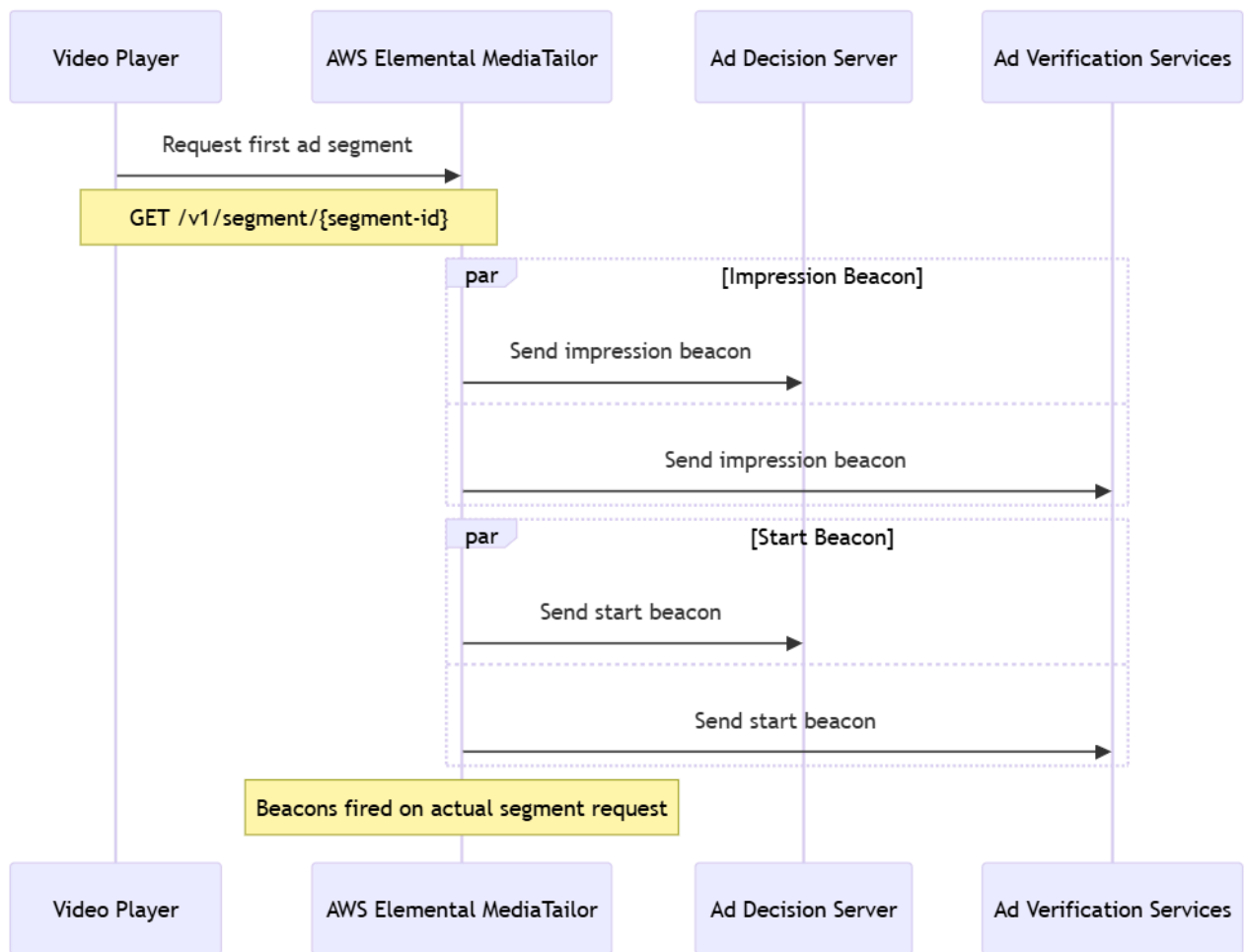
Phase 1: Session initialization

The player requests a manifest from MediaTailor, which returns a personalized manifest containing ad segment URLs:



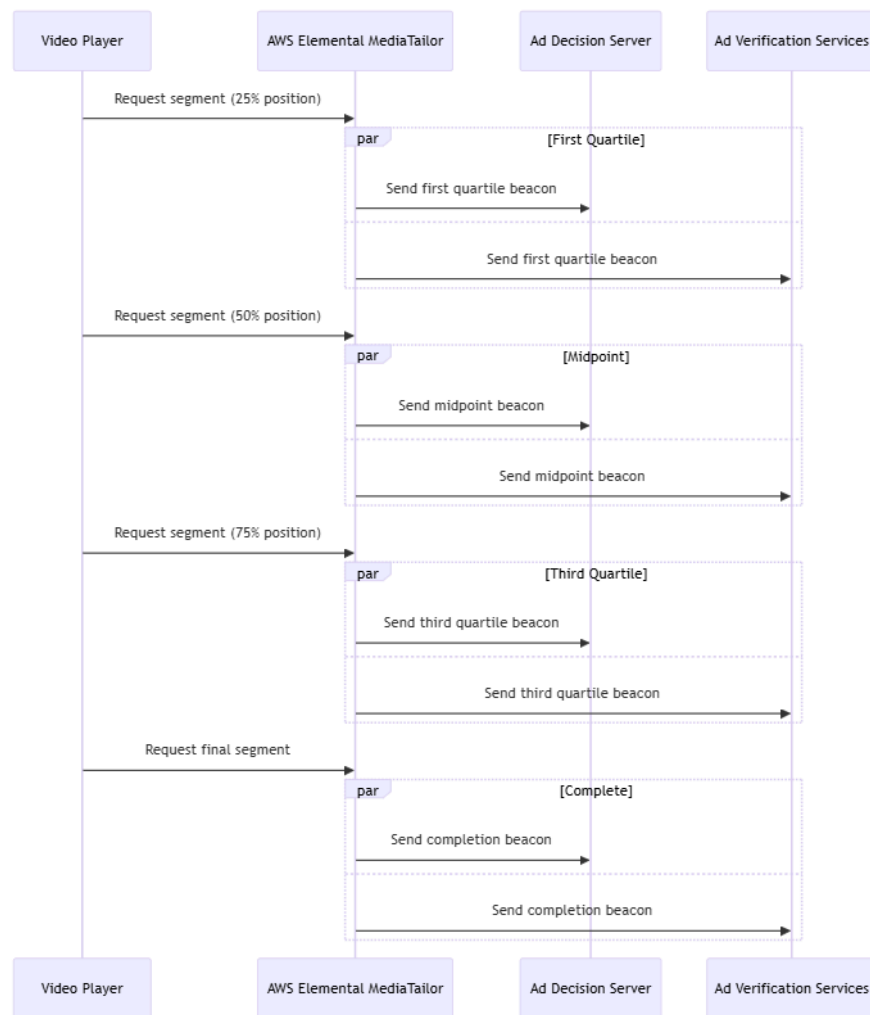
Phase 2: Ad request and impression tracking

When the player requests the first ad segment, MediaTailor fires impression and start beacons to both the Ad Decision Server and Ad Verification Services:



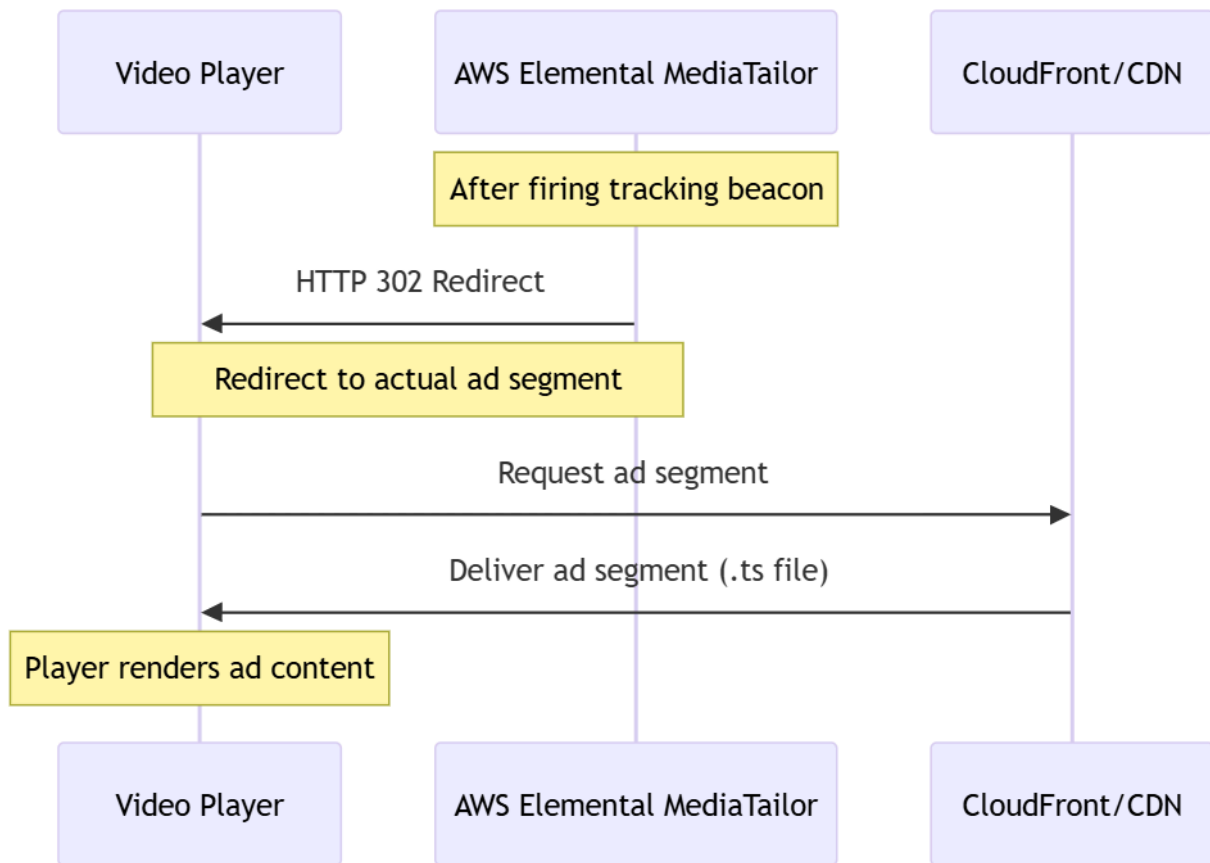
Phase 3: Quartile tracking

MediaTailor fires quartile beacons (first quartile, midpoint, third quartile, completion) based on subsequent segment requests:



Phase 4: Segment delivery

After firing tracking beacons, MediaTailor redirects to the actual ad segment from Amazon CloudFront or your CDN:



The server-side tracking workflow includes the following key timing behaviors:

1. **Session initialization** - The player requests a manifest from MediaTailor. MediaTailor returns a personalized manifest containing ad segment URLs with the /v1/segment path.
2. **Manifest parsing and caching** - The player parses the manifest and may pre-load or cache segment information. **No tracking events are fired during this phase**, regardless of player caching behavior.
3. **Ad segment request and impression tracking** - When the player actually requests the first ad segment (typically for playback), MediaTailor fires the impression beacon and start tracking event to both the Ad Decision Server and Ad Verification Services. This occurs on the actual HTTP request to the /v1/segment URL, not when the manifest is parsed.
4. **Quartile tracking based on segment requests** - MediaTailor fires quartile beacons (first quartile, midpoint, third quartile, completion) to both the Ad Decision Server and Ad Verification Services.

based on subsequent segment requests that correspond to the calculated quartile positions within the ad duration.

5. **Segment delivery** - After firing the appropriate tracking beacon, MediaTailor issues an HTTP redirect to the actual ad segment (either from Amazon CloudFront or your CDN).

Player caching and pre-loading considerations

MediaTailor server-side tracking is designed to be compatible with various player caching and pre-loading strategies while maintaining accurate impression measurement:

- **Manifest pre-loading** - Players that pre-load or cache manifest information do not trigger tracking events. Tracking events are only fired when actual segment requests are made.
- **Segment pre-fetching** - If a player pre-fetches ad segments before playback, tracking events will fire when those segments are requested, potentially earlier than the actual playback time. This behavior aligns with industry standards that consider segment requests as valid impressions.
- **Player buffering** - Standard player buffering behavior (requesting segments slightly ahead of playback) will trigger tracking events at the appropriate times based on the segment request pattern.

Troubleshooting tracking discrepancies

If you notice discrepancies between MediaTailor server-side tracking and third-party metrics, consider the following factors:

- **Player behavior differences** - Different players may have varying pre-fetching and buffering strategies that affect when segment requests are made.
- **Network conditions** - Poor network conditions may cause players to request segments multiple times or at different intervals than expected.
- **CDN configuration** - Incorrect CDN caching of `/v1/segment` requests can lead to missed or duplicate tracking events.
- **Session management** - Ensure that each playback session uses a unique session identifier to prevent tracking event conflicts.

For detailed troubleshooting guidance, see [Troubleshooting common issues](#).

MediaTailor server-side tracking features and capabilities

AWS Elemental MediaTailor automatically applies these integrated server-side tracking features to optimize ad measurement accuracy and reliability. The system prevents duplicate beacons, manages traffic during high-volume periods, maintains proper event sequencing, and provides comprehensive performance monitoring without requiring any configuration from you. You only need to ensure your ad decision server (ADS) provides the tracking beacons in the VAST response.

Note

These features are available for new customers starting September 30, 2025. Existing customers will have access throughout 2025 as part of ongoing service improvements. If you want immediate access to these features, contact [AWS Support](#).

Note

These features apply to both stitched (SSAI) and server-guided (SGAI) ad insertion methods. The beacon types and timing are the same in both modes. They differ in how MediaTailor triggers beacons — see [Server-side tracking with server-guided ad insertion \(SGAI\)](#) for details on SGAI server-side beaconing.

Beacon deduplication

MediaTailor prevents duplicate beacon firing for identical ad events. The server-side tracking system sends each impression, quartile, and completion beacon only once per ad viewing session. When video players request the same ad segment multiple times due to network conditions, bitrate changes, or buffering strategies, MediaTailor tracks fired beacons and blocks redundant transmissions.

Deduplication automatically resolves common scenarios that cause inflated beacon counts:

- **Adaptive bitrate streaming** - When players download different quality variants of the same ad segment
- **Network retry scenarios** - When players re-request segments due to network issues or timeouts
- **Player buffering strategies** - When players pre-fetch or re-fetch segments for buffering purposes

The system is designed to fire impression beacons only once, even when players switch between different quality levels.

Adaptive throttling and beacon retries

MediaTailor automatically manages beacon traffic rates based on server response indicators. The system monitors HTTP response patterns, connection timeouts, and error codes to detect congestion, then adjusts traffic rates accordingly. When the system identifies server stress indicators, it reduces traffic rates for the affected domain and automatically increases rates when servers demonstrate improved capacity.

The system monitors server health using these indicators:

- **HTTP connection timeouts** - When measurement platforms don't respond within expected timeframes
- **Error response codes** - 503, 504, and 507 responses that indicate server overload. Your ad server must also support these error codes for full compatibility.
- **Response patterns** - Measurement platform performance changes that indicate capacity issues

Retry behavior automatically attempts delivery for up to 1 hour with minimum 30-second delays between attempts. This retry behavior cannot be configured.

Beacon traffic per second management

You can set TPS limits to control beacon delivery rates. This is the only configurable setting for server-side tracking features. Account-level limits cap the total number of ad tracking requests sent across all measurement partners. MediaTailor enforces a minimum TPS limit of 10,000 to provide sufficient capacity for enterprise-scale operations.

Submit an AWS support ticket to establish TPS limits with the following information:

- **AWS account ID** - Your specific account identifier
- **Target region** - The AWS region where you want the TPS limit applied
- **Desired TPS threshold** - Your required transactions per second limit (minimum 10,000)

By default, there is no TPS limit. You can request a TPS limit if your ad decision server (ADS) requires it, but the limit must be greater than 10,000 TPS. MediaTailor will not exceed your

specified limit, but does not guarantee consistent throughput up to that limit. Your ad decision server will tell you what TPS limits it can support.

In-order beaconing

MediaTailor automatically maintains sequential delivery of ad tracking events. The system preserves beacon ordering even when network issues, retries, or traffic management occur. This ensures measurement partners receive events in the correct order for accurate analytics.

The system follows the standard industry beacon sequence:

1. **Start events** - Fire when ad playback begins
2. **First quartile events** - Fire at 25% ad completion
3. **Midpoint events** - Fire at 50% ad completion
4. **Third quartile events** - Fire at 75% ad completion
5. **Completion events** - Fire when ads finish

These features work together automatically:

- Beacons are held during throttling to maintain proper order
- Each measurement partner domain has separate event queues to prevent disruption during rate adjustments
- Deduplication tracks event type and timeline position while maintaining chronological order

Client-side ad tracking

Using the AWS Elemental MediaTailor client-side tracking API, you can incorporate player controls during ad breaks in streaming workflows. In client-side tracking, the player or client emits tracking events, such as impression and quartile ad beaconing, to the Ad Decision Server (ADS) and other ad-verification entities. These events track both the overall ad break status and the individual ad avails within each break. For more information about impression and quartile (ADS) and other ad-verification entities. For more information about impression and quartile ad beaconing, see [Client-side beaconing](#). For more information about ADS and other ad-verification entities, see [Client-side ad-tracking integrations](#).

For information about passing player parameters and session data to the ADS for client-side tracking, see [MediaTailor player variables for ADS requests](#) and [MediaTailor session variables for ADS requests](#).

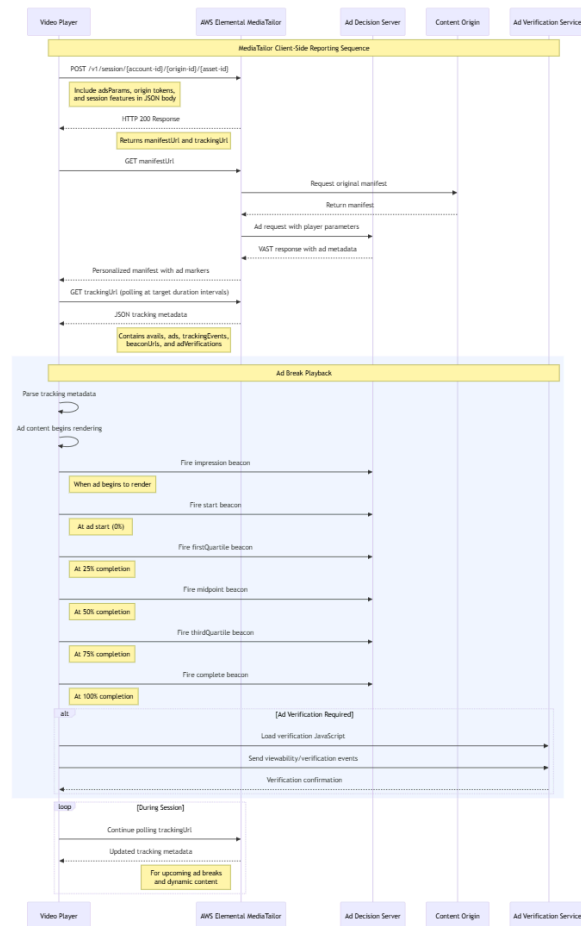
Client-side tracking enables functionality like the following:

- Ad-break countdown timers - For more information, see [Ad countdown timer](#).
- Ad click-through - For more information, see [Ad click-through](#).
- Display of companion ads - For more information, see [Companion ads](#).
- Skippable ads - For more information, see [Skippable ads](#).
- Display of VAST icons for privacy compliance - For more information, see [Icons for Google Why This Ad \(WTA\)](#).
- Control of player scrubbing during ads - For more information, see [Scrubbing](#).

Using the MediaTailor client-side tracking API, you can send metadata to the playback device that enables functionality in addition to client-side tracking:

Client-side reporting workflow

The following diagram shows the complete client-side reporting workflow from session initialization through ad playback and beaconing:



The client-side reporting workflow includes the following steps:

1. **Session initialization** - The video player sends a POST request to the MediaTailor session endpoint with JSON metadata including `adParams`, origin tokens, and session features. MediaTailor responds with `manifestUrl` and `trackingUrl` for the session.
2. **Manifest request and ad decision** - The player requests the personalized manifest from MediaTailor. MediaTailor requests the original content manifest from the origin, makes an ad request to the Ad Decision Server (ADS) using player parameters, receives a VAST response with ad metadata, and delivers a personalized manifest with ad markers to the player.
3. **Tracking data retrieval** - The player polls the tracking URL at regular intervals (matching target duration for HLS or minimum update period for DASH). MediaTailor returns JSON tracking metadata containing avails, ads, tracking events, beacon URLs, and ad verification data.
4. **Ad playback and beacons** - During ad breaks, the player parses the tracking metadata, fires impression beacons when ads begin rendering, fires quartile beacons (start, firstQuartile, midpoint, thirdQuartile, complete) at appropriate timing, loads and executes ad verification

JavaScript if required, and sends viewability/verification events to third-party verification services.

5. **Continuous polling** - The player continues polling the tracking URL throughout the session to receive updated metadata for upcoming ad breaks and dynamic content.

This workflow enables advanced features such as ad countdown timers, click-through functionality, companion ads, skippable ads, and VAST icon display for privacy compliance.

Topics

- [Enabling client-side tracking](#)
- [Ad server parameters](#)
- [Origin interaction query parameters](#)
- [Session-configured features](#)
- [Best practices for client-side tracking](#)
- [Client-side ad-tracking schema and properties](#)
- [Ad-tracking activity timing](#)
- [Player controls and functionality for client-side ad tracking](#)
- [Client-side beaconing](#)
- [Hybrid mode with server-side ad beacons](#)
- [Client-side ad-tracking integrations](#)
- [Paging through ad beacons with GetTracking](#)

Enabling client-side tracking

You enable client-side tracking for each session. The player makes an HTTP POST to the MediaTailor configuration's session-initialization prefix endpoint. Optionally, the player can send additional metadata for MediaTailor to use when making ad calls, calling the origin for a manifest, and invoking or disabling MediaTailor features at the session level.

The following example shows the structure of the JSON metadata:

```
{
  "adsParams": {
    "param1": "value1",
    # 'adsParams' is case sensitive
    # key is not case sensitive
```

```
    "param2": "value2",           # Values can contain spaces. For example, 'value 2'
    is an allowed value.
  },
  "origin_access_token":"abc123", # this is an example of a query parameter designated
  for the origin
  "overlayAvails":"on"           # 'overlayAvails' is case sensitive. This is an
  example of a feature that is enabled at the session level.
}
```

Use the MediaTailor console or API to configure the ADS request template URL to reference these parameters. In the following example, `player_params.param1` are the player parameters for `param1`, and `player_params.param2` are the player parameters for `param2`.

```
https://my.ads.com/path?param1=[player_params.param1]&param2=[player_params.param2]
```

Ad server parameters

At the topmost level of the JSON structure is an `adsParams` JSON object. Inside this object are key/value pairs that MediaTailor can read and send to the ad server in all session requests. MediaTailor supports the following ad servers:

- Google Ad Manager
- SpringServe
- FreeWheel
- Publica

Origin interaction query parameters

Any reserved key/value pairs within the topmost level of the JSON structure, such as `adsParams`, `availSuppression`, and `overlayAvails`, aren't added to the origin request URL in the form of query parameters. Every session manifest request that MediaTailor makes to the origin contains these query parameters. The origin ignores extraneous query parameters. For example, MediaTailor can use the key/value pairs to send access tokens to the origin.

Session-configured features

Use the session-initialization JSON structure to enable, disable, or override MediaTailor features such as `overlayAvails`, `availSuppression`, and `adSignaling`. Any feature configurations passed during session initialization override the setting at the MediaTailor configuration level.

Note

The metadata submitted to MediaTailor at session initialization is immutable, and additional metadata cannot be added for the duration of the session. Use SCTE-35 markers to carry data that changes during the session. For more information, see [MediaTailor session variables for ADS requests](#).

Example: Performing client-side ad tracking for HLS

```
POST mediatailorURL/v1/session/hashed-account-id/origin-id/asset-id.m3u8

{
  "adsParams": {
    "deviceType": "ipad"    # This value does not change during the session.
    "uid": "abdgfdyei-2283004-ueu"
  }
}
```

Example: Performing client-side ad tracking for DASH

```
POST mediatailorURL/v1/session/hashed-account-id/origin-id/asset-id.mpd

{
  "adsParams": {
    "deviceType": "androidmobile",
    "uid": "xjhddli-9189901-uic"
  }
}
```

Reporting mode parameter

You can specify the reporting mode when initializing a session by including the `reportingMode` parameter in the request body. This parameter controls whether MediaTailor performs client-side or server-side ad tracking for the session.

- **client** - The player performs ad tracking and sends beacons to the ad server. This is the default mode if no `reportingMode` is specified.
- **server** - MediaTailor performs server-side ad tracking and sends beacons directly to the ad server.

Example Session initialization with server-side reporting mode

```
POST mediatailorURL/v1/session/hashed-account-id/origin-id/asset-id.m3u8

{
  "adsParams": {
    "deviceType": "ipad",
    "uid": "abdgfdyei-2283004-ueu"
  },
  "reportingMode": "server"
}
```

Example Session initialization with client-side reporting mode (explicit)

```
POST mediatailorURL/v1/session/hashed-account-id/origin-id/asset-id.mpd

{
  "adsParams": {
    "deviceType": "androidmobile",
    "uid": "xjhddli-9189901-uic"
  },
  "reportingMode": "client"
}
```

Note

The `reportingMode` parameter is set at session initialization and cannot be changed during the session. If no `reportingMode` is specified, MediaTailor defaults to client-side reporting to maintain backward compatibility.

A successful response is an HTTP 200 with a response body. The body contains a `manifestUrl` and a `trackingUrl` key. The values are relative URLs that the player can use for both playback and ad-event tracking purposes.

```
{
  "manifestUrl": "/v1/dashmaster/hashed-account-id/origin-id/asset-id.m3u8?aws.sessionId=session-id",
  "trackingUrl": "/v1/tracking/hashed-account-id/origin-id/session-id"
}
```

For more information on the client-side tracking schema, see [Client-side ad-tracking schema and properties](#).

Best practices for client-side tracking

This section outlines the best practices for client-side tracking in MediaTailor for both live and VOD workflows.

Live workflows

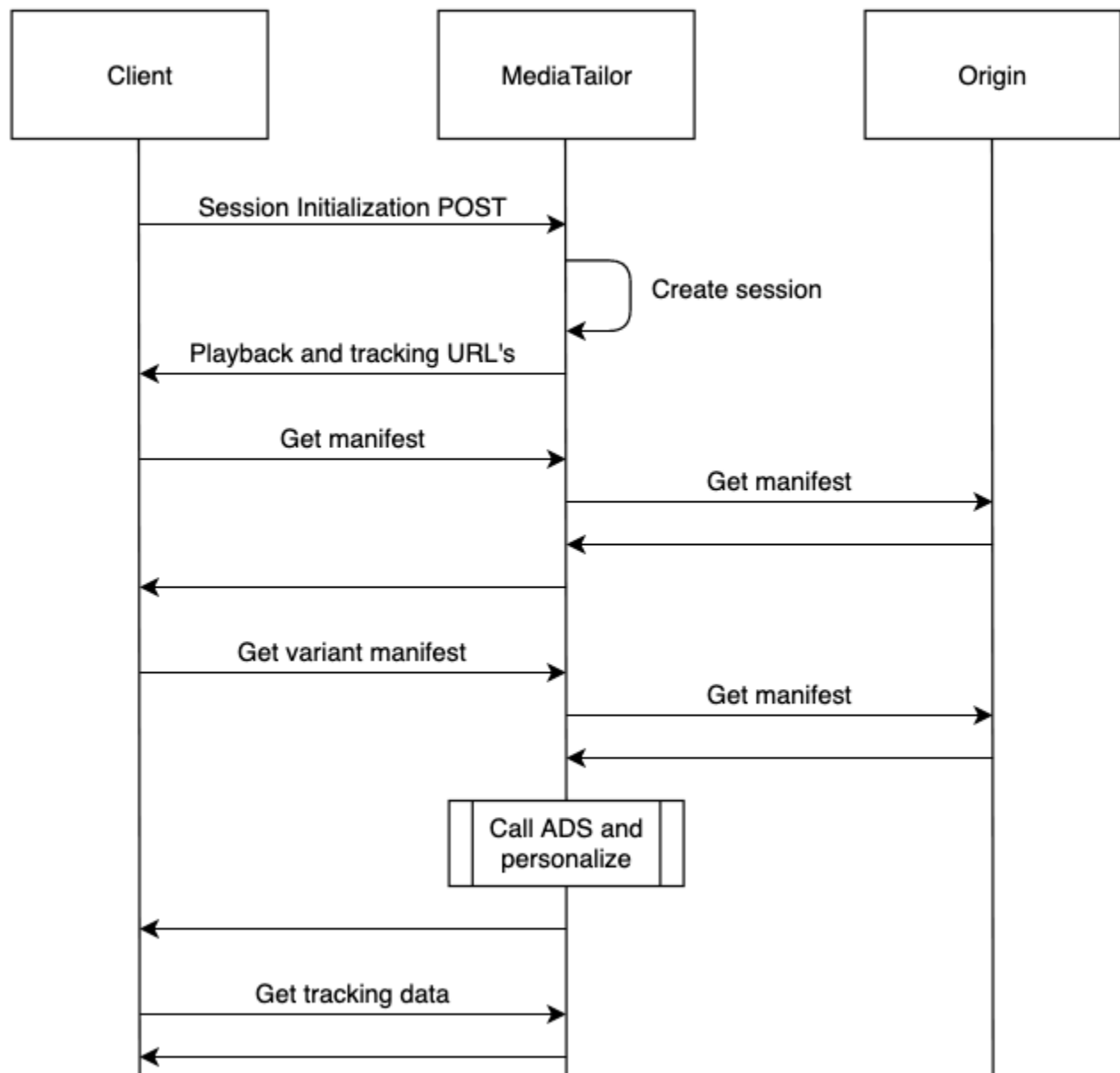
Poll the tracking endpoint at an interval matching every target duration for HLS, or minimum update period for DASH, in order to always have the most current ad-tracking metadata. Matching this interval is especially important in workflows where the creatives might have an interactive or overlay component.

Note

Some players support event listeners, which could be used as an alternative to polling. For example, the MediaTailor ad ID decoration feature would need to be enabled for each session. For more information, see [Ad ID decoration](#). Using this feature puts a date range (HLS) or event element (DASH) identifier over each ad in the avail. Players can use these manifest tags as a prompt to call the MediaTailor tracking endpoint for the session.

VOD workflows

Following a successful session initialization, and after MediaTailor receives the first manifest containing media, you only need to call the tracking endpoint once.



Server-guided ad insertion

Server-guided ad insertion (SGAI) sessions do not use the `GetTracking` API. Instead, when you use `aws.reportingMode=CLIENT`, MediaTailor provides tracking information in the `TRACKING` section of each asset list response when players request ad content. The session initialization response does not include a `trackingUrl`.

The asset list response for client-side tracked SGAI sessions has the following structure:

```
{
```

```
"ASSETS": [
  { "DURATION": 20.0, "URI": "https://cdn.example.com/ad1/master.m3u8" },
  { "DURATION": 10.0, "URI": "https://cdn.example.com/ad2/master.m3u8" }
],
"TRACKING": {
  ...VAST tracking events and beacon URLs for each ad...
}
}
```

When implementing client-side tracking for SGAI methods:

- Parse the TRACKING section from asset list responses rather than calling `GetTracking`
- Use the tracking URLs provided in the asset list for ad event reporting
- Fire tracking beacons based on actual ad playback events in the player
- Handle tracking for each ad break independently as asset lists are fetched

Important

The TRACKING section is only included in the asset list when `aws.reportingMode=CLIENT` is set. When server-side reporting is used (the default for SGAI), MediaTailor omits the TRACKING section and embeds beacon data in the ad URIs instead. For details, see [Server-side tracking with server-guided ad insertion \(SGAI\)](#).

Client-side ad-tracking schema and properties

With the MediaTailor client-side ad-tracking feature, you can integrate detailed client-side ad tracking data into your player environment. The following sections cover the overall ad-tracking schema, as well as the specific properties and values that make up the schema.

Schema

The following JSON structure shows the MediaTailor client-side ad-tracking schema. This representation illustrates the nesting structure of the schema to help you understand the relationships between different parts.

For detailed information about each property, see [the section called “Properties”](#).

```
{
```

```

"avails": [
  {
    "ads": [
      {
        "adID": "string",
        "adParameters": "string",
        "adSystem": "string",
        "adTitle": "string",
        "adVerifications": [
          {
            "executableResource": [
              {
                "apiFramework": "string",
                "type": "string",
                "uri": "string",
                "language": "string"
              }
            ],
            "javaScriptResource": [
              {
                "apiFramework": "string",
                "browserOptional": "string",
                "uri": "string"
              }
            ],
            "trackingEvents": [
              {
                "event": "string",
                "uri": "string"
              }
            ],
            "vendor": "string",
            "verificationParameters": "string"
          }
        ],
        "companionAds": [
          {
            "adParameters": "string",
            "altText": "string",
            "attributes": {
              "adSlotId": "string",
              "apiFramework": "string",
              "assetHeight": "string",
              "assetWidth": "string",

```

```

        "expandedHeight": "string",
        "expandedWidth": "string",
        "height": "string",
        "id": "string",
        "pxratio": "string",
        "renderingMode": "string",
        "width": "string"
    },
    "companionClickThrough": "string",
    "companionClickTracking": "string",
    "htmlResource": "string",
    "iFrameResource": "string",
    "sequence": "string",
    "staticResource": "string",
    "trackingEvents": [
        {
            "event": "string",
            "uri": "string"
        }
    ]
    },
    ],
    "creativeId": "string",
    "creativeSequence": "string",
    "duration": "string",
    "durationInSeconds": number,
    "extensions": [
        {
            "type": "string",
            "content": "string"
        }
    ],
    "icons": [
        {
            "attributes": {
                "apiFramework": "string",
                "duration": "string",
                "height": "string",
                "offset": "string",
                "program": "string",
                "pxratio": "string",
                "width": "string",
                "xPosition": "string",
                "yPosition": "string"
            }
        }
    ]
}

```

```

    },
    "htmlResource": "string",
    "iconClicks": {
      "iconClickThrough": "string",
      "iconClickTracking": {
        "id": "string"
      }
    },
    "iconClickFallbackImages": [
      {
        "altText": "string",
        "height": "string",
        "width": "string",
        "staticResource": {
          "creativeType": "string",
          "uri": "string"
        }
      }
    ]
  },
  "iconViewTracking": "string",
  "iFrameResource": "string",
  "staticResource": {
    "creativeType": "string",
    "uri": "string"
  }
}
],
"mediaFiles": {
  "adParameters": "string",
  "duration": "string",
  "durationInSeconds": number,
  "mediaFilesList": [
    {
      "apiFramework": "string",
      "delivery": "string",
      "height": "string",
      "maintainAspectRatio": "string",
      "mediaFileUri": "string",
      "mediaType": "string",
      "scalable": "string",
      "width": "string",
      "bitrate": "string"
    }
  ]
},

```

```

    "mezzanine": "string",
    "startTime": "string",
    "startTimeInSeconds": number,
    "trackingEvents": [
      {
        "beaconUrls": ["string"],
        "duration": "string",
        "durationInSeconds": number,
        "dateTime": "string",
        "eventId": "string",
        "eventType": "string",
        "startTime": "string",
        "startTimeInSeconds": number
      }
    ],
    "startTime": "string",
    "startTimeInSeconds": number,
    "dateTime": "string",
    "adBreakTrackingEvents": [...],
    "vastAdId": "string"
  },
  "adType": "string",
  "availID": "string",
  "duration": "string",
  "durationInSeconds": number,
  "startTime": "string",
  "startTimeInSeconds": number,
  "dateTime": "string",
  "adMarkerDuration": "string",
  "adProgramDateTime": "string",
  "dashAvailabilityStartTime": "string",
  "hlsAnchorMediaSequenceNumber": "string"
},
"nonLinearAvails": [
  {
    "nonLinearAds": [...],
    "nonLinearAdsList": [...]
  }
],
"nextToken": "string",
"meta": {}

```

}

Properties

The following table lists the properties in the client-side tracking API, their definitions, value types, and examples.

Property	Definition	Value type	Example
adID	- HLS - the sequence number associated with the beginning of the ad - DASH - the period ID of the ad Path: /avails/ads/adId VAST mapping: None	String	10
adBreakTrackingEvents	An array that carries VMAP tracking events from the VAST response. For more information, see section 2.3.3 of the VMAP 1.0 specification. Path: /avails/ads/adBreakTrackingEvents	Array	[]
adMarkerDuration	The avail duration observed from the ad marker in the manifest.	String	30
adParameters	A string of ad parameters, from the VAST VPAID, that MediaTailor passes to the player. Path: /avails/ads/adParameters VAST mapping: VAST/Ad/InLine/Creatives/Cr	String	

Property	Definition	Value type	Example
	eatative/Linear/AdParameters		
adProgram DateTime	- HLS - the date, in ISO/IEC 8601:2004 format, that represents the first media sequence of the ad. - DASH - [Not specified]	String	
ads	An array containing the ad objects that make up the avail. The ads are listed in the order they appear in the manifest. Path: /avails/ads	Array	[]
adSystem	The name of the system that serves the ad.  Important Make sure to provide a value. If you don't provide a value, issues can arise.	String	myADS
adTitle	The title of the ad.	String	ad1

Property	Definition	Value type	Example
adVerifications	Contains the resources and metadata required to execute third-party measurement code in order to verify creative playback. For more information about this property, see section 3.16 of the VAST 4.2 specification. MediaTailor supports adVerifications as VAST 3 extension nodes. Path: /avails/ads/adVerifications VAST mapping: VAST/Ad/Inline/AdVerifications	Array	[]
altText	The alternative text for an image of a companion ad. This text enables players with descriptive-audio support for the visually impaired to read back a description of the image. Path: /avails/ads/companionAds/altText	String	video sequence advertising sneakers
apiFramework	Set to VPAID to tell the player that this ad is a VPAID ad. Can appear in multiple locations in the schema.	String	VPAID

Property	Definition	Value type	Example
<code>availID</code>	- HLS - the sequence number associated with the start of the ad avail. - DASH - the period ID of the ad avail, which is usually the period ID of the content that is to be replaced with an ad. Path: <code>/avails/availID</code>	String	34 - PT34S_1
<code>avails</code>	An array containing ad-break objects, or <i>avails</i>, that are presented in the active manifest window. The avails are listed in the order they appear in the manifest. Path: <code>/avails</code>	Array	<code>[]</code>
<code>adType</code>	The type of the ad. Path: <code>/avails/adType</code> and <code>/avails/ads/adType</code>	String	
<code>dateTime</code>	Program date time, in ISO 8601 seconds format, for the start of the ad avail or ad. Path: <code>/avails/dateTime</code> and <code>/avails/ads/dateTime</code>	String	

Property	Definition	Value type	Example
duration	Length, in ISO 8601 seconds format. The response includes durations for the entire ad avail and for each ad and beacon, though beacon durations are always zero. Path: /avails/duration and /avails/ads/duration	String	15.015
durationInSeconds	Length, in seconds format. Path: /avails/durationInSeconds and /avails/ads/durationInSeconds	Number	
extensions	Custom extensions of VAST that ad servers use. For more information about extensions, see section 3.18 of the VAST 4.2 specification. Path: /avails/ads/extensions VAST mapping: VAST/Ad/InLine/Extensions	Array	[]
icons	Icon elements for the ad. Path: /avails/ads/icons VAST mapping: VAST/Ad/InLine/Creatives/Creative/Linear/Icons	Array	

Property	Definition	Value type	Example
<code>mediaFiles</code>	Video and other assets that the player needs for the ad avail. Path: <code>/avails/ads/mediaFiles</code>	Object	
<code>nonLinearAvails</code>	Array of non-linear ad avail objects. Path: <code>/nonLinearAvails</code>	Array	
<code>executableResource</code>	Executable resources for verification. Path: <code>/avails/ads/adVerifications/executableResource</code> VAST mapping: <code>VAST/Ad/Inline/AdVerifications/Verification/ExecutableResource</code>	Array	
<code>javascriptResource</code>	JavaScript resources for verification. Path: <code>/avails/ads/adVerifications/javascriptResource</code> VAST mapping: <code>VAST/Ad/Inline/AdVerifications/Verification/JavaScriptResource</code>	Array	

Property	Definition	Value type	Example
trackingEvents	Tracking events for verification or companion ads. Path: /avails/ads/adVerifications/trackingEvents or /avails/ads/companionAds/trackingEvents	Array	
vendor	Verification vendor. Path: /avails/ads/adVerifications/vendor VAST mapping: VAST/Ad/InLine/AdVerifications/Verification/@vendor	String	
uri	URI that points to either an executable asset, a video asset, or a tracking endpoint. Path: Various locations in the schema VAST mapping: Various CDATA elements in VAST	String	https://tracking.example.com/impression
verificationParameters	Verification parameters. Path: /avails/ads/adVerifications/verificationParameters VAST mapping: VAST/Ad/InLine/AdVerifications/Verification/VerificationParameters	String	

Property	Definition	Value type	Example
attributes	Companion ad attributes like dimensions and rendering mode. Path: /avails/ads/companionAds/attributes	Object	
companionClickThrough	A URL to the advertiser's page that the media player opens when the viewer clicks the companion ad. Path: /avails/ads/companionAds/companionClickThrough VAST mapping: VAST/Ad/InLine/Creatives/Creative/CompanionAds/Companion/CompanionClickThrough	String	https://aws.amazon.com/
companionClickTracking	The tracking URL for the companionClickThrough property. Path: /avails/ads/companionAds/companionClickTracking VAST mapping: VAST/Ad/InLine/Creatives/Creative/CompanionAds/Companion/CompanionClickTracking	String	https://myads.com/beaconing/event=clicktracking

Property	Definition	Value type	Example
htmlResource	The CDATA-encoded HTML that's inserted directly within the streaming provider's HTML page. Path: /avails/ads/companionAds/htmlResource VAST mapping: VAST/Ad/InLine/Creatives/Creative/CompanionAds/Companion/HTMLResource	String	<pre><![CDATA[<!doctype html><htm l><head>< meta name= \"viewport \" content= \"width=1, initial-s cale=1.0, minimum-s cale=1.0, ...]]></pre>
iFrameResource	The URL to an HTML resource file that the streaming provider loads into an iframe. Path: /avails/ads/companionAds/iFrameResource VAST mapping: VAST/Ad/InLine/Creatives/Creative/CompanionAds/Companion/iFrameResource	String	
sequence	The sequence value specified for the creative in the VAST response. Path: /avails/ads/companionAds/sequence	String	1

Property	Definition	Value type	Example
<code>startTime</code>	The time position, in ISO 8601 seconds format. For HLS, this is relative to the beginning of the playback session. For DASH, this is relative to the manifest's AST (Availability Start Time). The response includes start times for the entire ad avail and for each ad and beacon. Path: <code>/avails/startTime</code> and <code>/avails/ads/startTime</code>	String	PT18.581355S
<code>startTimeInSeconds</code>	The time position, in seconds format. For HLS, this is relative to the beginning of the playback session. For DASH, this is relative to the manifest's AST (Availability Start Time). The response includes start times for the entire ad avail and for each ad and beacon. Path: <code>/avails/startTimeInSeconds</code> and <code>/avails/ads/startTimeInSeconds</code>	Number	18.581
<code>eventId</code>	- HLS - the sequence number associated with the beacon. - DASH - the <code>ptsTime</code> of the start of the ad.	String	

Property	Definition	Value type	Example
event	The name of the tracking event. Path: /avails/ads/adVerifications/trackingEvents/event or /avails/ads/companionAds/trackingEvents/event	String	impression , start, firstQuartile , midpoint, thirdQuartile , complete
beaconUrls	The URL where MediaTailor sends the ad beacon. Path: /avails/ads/mediaFiles/trackingEvents/beaconUrls	Array	
bitrate	The bitrate of the video asset. This property is not typically included for an executable asset.	String	2048
companion Ads	One or more companion ad-content specifications, each of which specifies a resource file to use. Companion ads accompany the ad avail and provide content, like a frame around the ad or a banner, to display near the video. Path: /avails/ads/companionAds	Array	[]
creativeId	The Id attribute value of the Creative tag for the ad.	String	creative-1

Property	Definition	Value type	Example
<code>creativeSequence</code>	The sequence in which an ad should play, according to the <code>Ad@id</code> value in the VAST response.	String	1
<code>dashAvailabilityStartTime</code>	For live/dynamic DASH, the <code>MPD@availabilityStartTime</code> of the origin manifest.	String	2022-10-05T19:38:39.263Z
<code>delivery</code>	Indicates whether a progressive or streaming protocol is being used.	String	progressive
<code>eventType</code>	The type of beacon. Path: <code>/avails/ads/mediaFiles/trackingEvents/eventType</code>	String	impression
<code>height</code>	The height, in pixels, of the video asset.	String	360
<code>hlsAnchorMediaSequenceNumber</code>	The media-sequence number of the first/oldest media sequence seen in the HLS origin manifest.	String	77
<code>maintainAspectRatio</code>	Indicates whether to maintain the video's aspect ratio while scaling.	Boolean	true
<code>mediaFilesList</code>	Specifies the video and other assets that the player needs for the ad avail. Path: <code>/avails/ads/mediaFiles/mediaFilesList</code>	Array	[]

Property	Definition	Value type	Example
mediaFileUri	URI that points to either an executable asset or a video asset.	String	https://myad.com/ad/ad134/vpaid.js
mediaType	The MIME type of the creative or companion asset.	String	video/mp4
meta	Additional metadata for the ad.	Object	
mezzanine	The URL of the mezzanine MP4 asset, specified if the VPAID ad includes one. Path: /avails/ads/mediaFiles/mezzanine	String	https://gcdn.2mdn.net/video/playback/id/itag/c/k2/file/file.mp4
nextToken	The value of the token that points to the next page of results, when such a value exists.	String	UFQz0S44NzNTXzIwMjMtMDctMzFUMTY6NTA6MDYuMzUwNjI2ODQ1Wl8x
nonLinearAds	Non-linear ads that appear alongside the video content.	Array	[]
nonLinearAdsList	List of non-linear ads.	Array	[]
scalable	Indicates whether to scale the video to other dimensions.	Boolean	true

Property	Definition	Value type	Example
skipOffset	The time value that identifies when the player makes skip controls available to the user.	String	00:00:05
staticResource	The URL to a static creative file that's used for the ad component. Path: /avails/ads/companionAds/staticResource	String	https://very-interactive-ads.com/campaign1/file.json?c=1019113602
vastAdId	The Id attribute value of the Ad tag.	String	ad1
width	The width, in pixels, of the video asset.	String	640
xPosition	The horizontal position of an icon within the video player. Can be a specific pixel value or a position like "left" or "right". Path: /avails/ads/icons/attributes/xPosition	String	left or 10
yPosition	The vertical position of an icon within the video player. Can be a specific pixel value or a position like "top" or "bottom". Path: /avails/ads/icons/attributes/yPosition	String	top or 10

Property	Definition	Value type	Example
<code>iconClicks</code>	Contains click-through and tracking information for an icon. Path: <code>/avails/ads/icons/iconClicks</code>	Object	
<code>iconClickThrough</code>	A URL to the advertiser's page that the media player opens when the viewer clicks the icon. Path: <code>/avails/ads/icons/iconClicks/iconClickThrough</code>	String	<code>https://advertiser.com/landing-page</code>
<code>iconClickTracking</code>	The tracking URL for the <code>iconClickThrough</code> property. Path: <code>/avails/ads/icons/iconClicks/iconClickTracking</code>	Object	
<code>iconClickFallbackImages</code>	An array of fallback images to display if the icon cannot be displayed. Path: <code>/avails/ads/icons/iconClicks/iconClickFallbackImages</code>	Array	
<code>iconViewTracking</code>	The URL for tracking when an icon is viewed. Path: <code>/avails/ads/icons/iconViewTracking</code>	String	<code>https://tracking.example.com/icon-view</code>

Property	Definition	Value type	Example
offset	The time offset for when an icon should appear during ad playback. Path: /avails/ads/icons/attributes/offset	String	00:00:05
program	The program or initiative associated with the icon, such as "AdChoices". Path: /avails/ads/icons/attributes/program	String	AdChoices
pxratio	The pixel ratio for the icon or companion ad, used for high-DPI displays. Path: /avails/ads/icons/attributes/pxratio or /avails/ads/companionAds/attributes/pxratio	String	1 or 2
type	The type of resource or extension. Path: /avails/ads/extensions/type or /avails/ads/adVerifications/executableResource/type	String	text/javascript
content	The content of an extension. Path: /avails/ads/extensions/content	String	

Property	Definition	Value type	Example
language	The programming language of an executable resource. Path: /avails/ads/adVerifications/executableResource/language	String	javascript
browserOptional	Indicates whether browser support is required for the JavaScript resource. Path: /avails/ads/adVerifications/javascriptResource/browserOptional	String	true or false
id	An identifier for various elements in the schema. Path: /avails/ads/companionAds/attributes/id or /avails/ads/icons/iconClicks/iconClickTracking/id	String	companion-1
assetHeight	The height of the companion ad asset. Path: /avails/ads/companionAds/attributes/assetHeight	String	250

Property	Definition	Value type	Example
assetWidth	The width of the companion ad asset. Path: /avails/ads/companionAds/attributes/assetWidth	String	300
expandedHeight	The height of the companion ad when expanded. Path: /avails/ads/companionAds/attributes/expandedHeight	String	600
expandedWidth	The width of the companion ad when expanded. Path: /avails/ads/companionAds/attributes/expandedWidth	String	600
renderingMode	The rendering mode for the companion ad. Path: /avails/ads/companionAds/attributes/renderingMode	String	default or transparent
adSlotId	The ID of the ad slot where the companion ad should be displayed. Path: /avails/ads/companionAds/attributes/adSlotId	String	banner-1

Property	Definition	Value type	Example
<code>creativeType</code>	The MIME type of the creative asset. Path: <code>/avails/ads/icons/staticResource/creativeType</code>	String	<code>image/png</code>

Ad-tracking activity timing

With client-side reporting, the player must emit tracking events (beacons) with a level of precision. Using the MediaTailor client-side tracking schema, you can ensure that, for every avail, ad, companion, overlay, and tracking events, timing and duration information is present, and in different forms.

Use the following MediaTailor key/value pairs for the player to accurately reconcile ad-event activities, such as tracking events, with playback position:

- [startTime](#)
- [startTimeInSeconds](#)
- [adProgramDateTime](#)
- [adID/eventId](#)

HLS and DASH implement the value of `startTime` and `startTimeInSeconds` differently:

- HLS - The `startTime` values are relative to the beginning of the playback session. The beginning of the playback session is defined as time zero. The ad's `startTime` is the sum of the cumulative values of all the EXT-INF segment durations leading up to the avail. The media-sequence number of the segment that the ad or tracking event falls on also corresponds to the `adId` or `eventId` in the client-side tracking response.
- DASH:
 - Live/dynamic manifests - The `startTime` values are relative to the `MPD@availabilityStartTime` of the DASH manifest. The `MPD@availabilityStartTime` is a timing anchor for all MediaTailor sessions that consume the stream.
 - VOD/static manifests - The `startTime` values are relative to the beginning of the playback session. The beginning of the playback session is defined as time zero. Each ad inside the avail

is contained inside its own Period element. The Period element has a `@start` attribute with a value that's the same as the `startTime` values in the client-side tracking payload. The `PeriodId` also corresponds to the `adId` or `eventId` in the client-side tracking response.

Example: HLS

In the following example, the MediaTailor session started, and the following manifest is the first one served to the client:

```
#EXTM3U
#EXT-X-VERSION:6
#EXT-X-TARGETDURATION:6
#EXT-X-MEDIA-SEQUENCE:4603263
#EXT-X-DISCONTINUITY-SEQUENCE:0
#EXT-X-PROGRAM-DATE-TIME:2023-05-03T21:24:23.295678Z
#EXTINF:4.010667,
https://123.cloudfront.net/out/v1/index_1_34.ts
#EXT-X-PROGRAM-DATE-TIME:2023-05-03T21:24:27.306345Z
#EXTINF:4.010667,
https://123.cloudfront.net/out/v1/index_1_35.ts
#EXT-X-PROGRAM-DATE-TIME:2023-05-03T21:24:31.317012Z
#EXTINF:4.010667,
https://123.cloudfront.net/out/v1/index_1_36.ts
#EXT-X-PROGRAM-DATE-TIME:2023-05-03T21:24:35.327679Z
#EXTINF:4.010667,
https://123.cloudfront.net/out/v1/index_1_37.ts
#EXT-X-PROGRAM-DATE-TIME:2023-05-03T21:24:39.338346Z
#EXTINF:2.538667,
https://123.cloudfront.net/out/v1/index_1_38.ts
#EXT-X-DISCONTINUITY
#EXT-X-KEY:METHOD=NONE
#EXT-X-PROGRAM-DATE-TIME:2023-05-03T21:24:41.453Z
#EXTINF:2.0,
https://123.cloudfront.net/tm/asset_1080_4_8_00001.ts
#EXT-X-PROGRAM-DATE-TIME:2023-05-03T21:24:43.453Z
#EXTINF:2.0,
https://123.cloudfront.net/tm/asset_1080_4_8_00002.ts
#EXT-X-PROGRAM-DATE-TIME:2023-05-03T21:24:45.453Z
#EXTINF:2.0,
https://123.cloudfront.net/tm/asset_1080_4_8_00003.ts
```

In the client-side tracking JSON payload, the following values apply:

- `startTime`: "PT18.581355S"
- `startTimeInSeconds`: 18.581
- `availProgramDateTime`: "2023-05-03T21:24:41.453Z"
- `adId`: 4603269

Example: DASH

In the following example, the MediaTailor session gets a midroll in the manifest. Note that the `@start` attribute value of the second period, which is the ad period, has a value that's relative to the `MPD@availabilityStartTime` value. This value is the one that MediaTailor writes into the client-side tracking response `startTime` fields, for all sessions.

```
<?xml version="1.0" encoding="UTF-8"?>
<MPD availabilityStartTime="2022-10-05T19:38:39.263Z" minBufferTime="PT10S"
  minimumUpdatePeriod="PT2S" profiles="urn:mpeg:dash:profile:isoff-live:2011"
  publishTime="2023-05-03T22:06:48.411Z" suggestedPresentationDelay="PT10S"
  timeShiftBufferDepth="PT1M30S" type="dynamic" xmlns="urn:mpeg:dash:schema:mpd:2011"
  xmlns:scte35="urn:scte:scte35:2013:xml" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
  instance" xsi:schemaLocation="urn:mpeg:dash:schema:mpd:2011 http://standards.iso.org/
  ittf/PubliclyAvailableStandards/MPEG-DASH_schema_files/DASH-MPD.xsd">
  <BaseURL>https://123.channel-assembly.mediatailor.us-west-2.amazonaws.com/v1/
  channel/my-channel/</BaseURL>
  <Location>https://123.cloudfront.net/v1/
  dash/94063eadf7d8c56e9e2edd84fdf897826a70d0df/MediaTailor-Live-HLS-DASH/channel/
  channel1/dash.mpd?aws.sessionId=794a15e0-2a7f-4941-a537-9d71627984e5</Location>
  <Period id="1683151479166_1" start="PT5042H25M59.903S"
  xmlns="urn:mpeg:dash:schema:mpd:2011">
    <BaseURL>https://123.cloudfront.net/out/v1/f1a946be8efa45b0931ea35c9055fb74/
    ddb73bf548a44551a0059c346226445a/eea5485198bf497284559efb8172425e/</BaseURL>
    <AdaptationSet ...>
      ...
    </AdaptationSet>
  </Period>
  <Period id="1683151599194_1_1" start="PT5042H27M59.931S">
    <BaseURL>https://123.cloudfront.net/
    tm/94063eadf7d8c56e9e2edd84fdf897826a70d0df/fpc5omz5wzd2rdepgieibp23ybyqyrme/</BaseURL>
    <AdaptationSet ...>
      ...
    </AdaptationSet>
  </Period>
```

```
</MPD>
```

In the client-side tracking JSON payload, the following values apply:

- `startTime`: "PT5042H27M59.931S"
- `startTimeInSeconds`: 18152879.931
- `availProgramDateTime`: *null*
- `adId`: 1683151599194_1_1

Player controls and functionality for client-side ad tracking

MediaTailor client-side tracking metadata supports various player controls and functionality. The following list describes popular player controls.

Topics

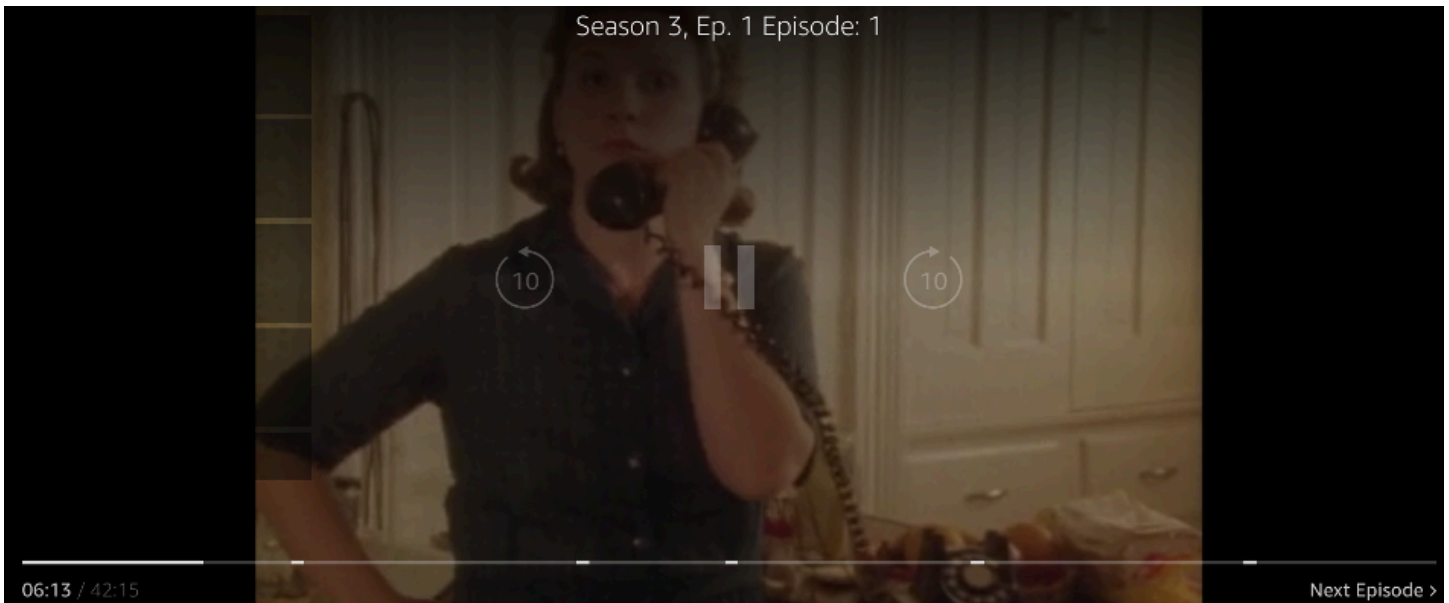
- [Scrubbing](#)
- [Ad countdown timer](#)
- [Skippable ads](#)
- [Ad click-through](#)
- [Companion ads](#)
- [Interactive ads \(SIMID\)](#)
- [Interactive ads \(VPAID\)](#)
- [Icons for Google Why This Ad \(WTA\)](#)

Scrubbing

To enhance the playback experience, the player can display ad positions in the playback timeline. MediaTailor makes these ad positions available in the form of `startTimeInSeconds` values in the client-side tracking response.

Note

Some streaming providers prevent scrubbing past an ad position.

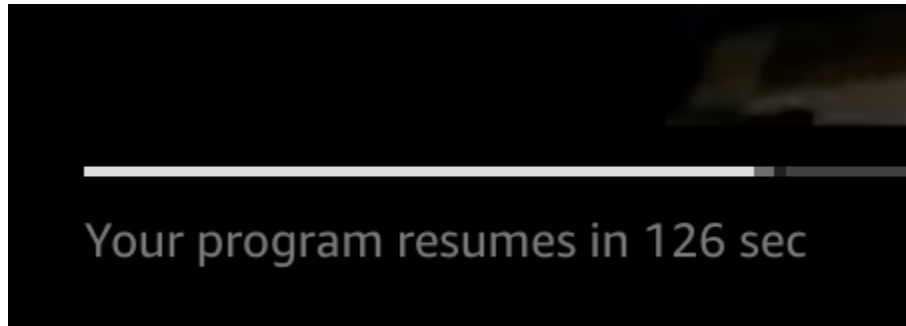


The following client-side tracking payload JSON response shows the avail (ad break) start time inside the root JSON object of the avails array. The player uses this data to show the location of the ad break on the player timeline, at 28 seconds.

```
{
  "avails": [
    {
      "adBreakTrackingEvents": [],
      "adMarkerDuration": null,
      "ads": [...],
      "availId": "7",
      "availProgramDateTime": null,
      "duration": "PT30S",
      "durationInSeconds": 30,
      "meta": null,
      "nonLinearAdsList": [],
      "startTime": "PT28S",
      "startTimeInSeconds": 28
    }
  ],
  "dashAvailabilityStartTime": null,
  "hlsAnchorMediaSequenceNumber": null,
  "nextToken": "UFQxMk0zNC44NjhTXzIwMjMtMDctMjFUMjA6MjM6MDcuNzc1NzE2MzAyWl8x",
  "nonLinearAvails": []
}
```

Ad countdown timer

With MediaTailor you can use an ad countdown timer to help keep your audience engaged during ad-break viewing. The audience can use the timer to understand when the ad break ends and their program resumes.



The elements in the client-side tracking metadata that play a role in the ad countdown timer are `startTime`, `startTimeInSeconds`, `duration`, and `durationInSeconds`. The player uses this metadata, along with the session's elapsed time that it tracks separately, to determine when to display the timer and the value it should be counting down from.

The following client-side tracking payload JSON response shows the information needed to display an ad countdown timer.

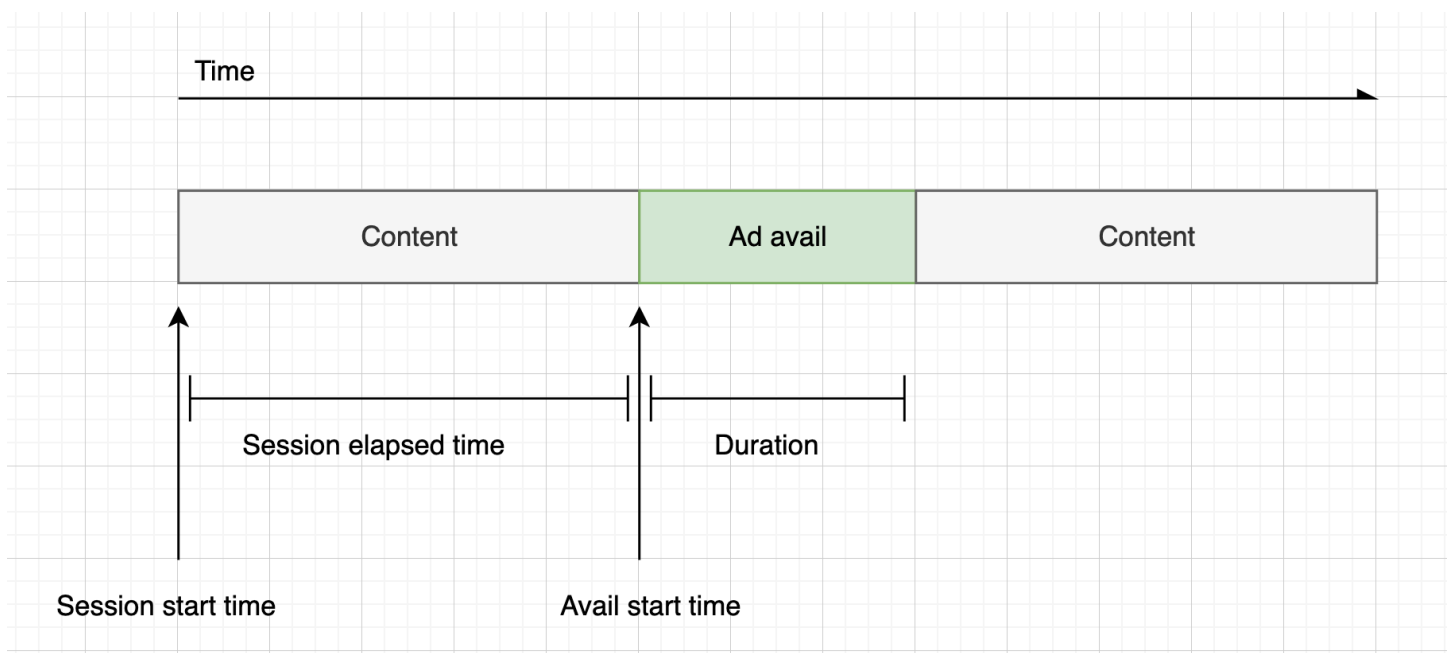
```
{
  "avails": [
    {
      "adBreakTrackingEvents": [],
      "adMarkerDuration": null,
      "ads": [...],
      "availId": "7",
      "availProgramDateTime": null,
      "duration": "PT30S",
      "durationInSeconds": 30,
      "meta": null,
      "nonLinearAdsList": [],
      "startTime": "PT28S",
      "startTimeInSeconds": 28
    }
  ],
  "dashAvailabilityStartTime": null,
  "hlsAnchorMediaSequenceNumber": null,
  "nextToken": "UFQxMk0zNC44NjhTXzIwMjMtMDctMjFUMjA6MjM6MDcuNzc1NzE2MzAyWl8x",
  "nonLinearAvails": []
}
```

}

When the session's elapsed time reaches the avail's start time, the player displays a countdown timer with a value that matches the avail's duration. The countdown-timer value decreases as the elapsed time progresses beyond the avail's start time.

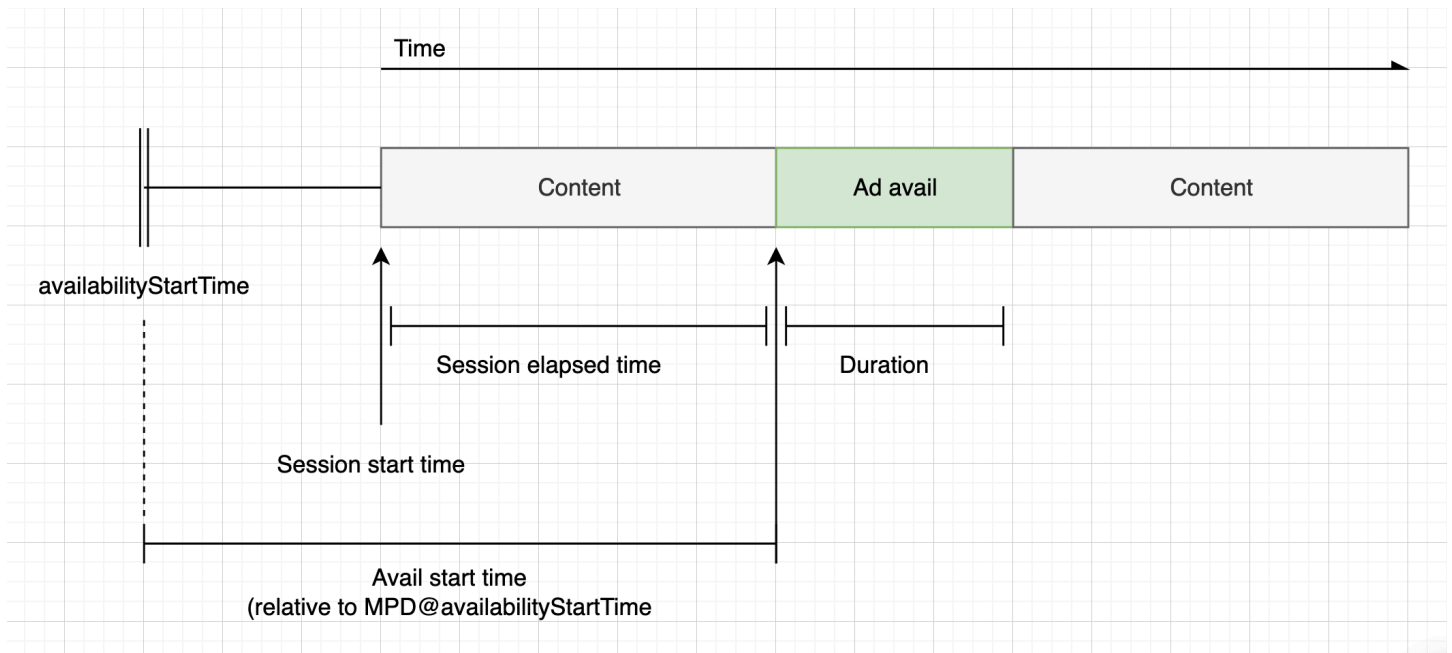
Example formula: Countdown timer for HLS (live and VOD) and DASH (VOD)

- $\text{session\_start\_time} = \text{the sum of all EXT-INF duration values} - \text{the duration value of the three newest EXT-INF media sequences}$
- $\text{timer value} = \text{duration} - (\text{session\_elapsed\_time} - \text{startTime})$



Example formula: Countdown timer for DASH (live)

- $\text{session\_start\_time} = (\text{newest segment's startTime} + \text{duration}) / \text{timescale} - \text{MPD@suggestedPresentationDelay}$
- $\text{timer value} = \text{duration} - (\text{session\_elapsed\_time} - \text{startTime})$



Skippable ads

Skippable ads are ad spots that allow the viewer to skip some of the ad to resume viewing the program. In VAST, the `Linear@skipOffset` attribute identifies a skippable ad.

The following VAST response shows how to use a skippable ad:

```
<?xml version="1.0" encoding="UTF-8"?>
<VAST xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="vast.xsd" version="3.0">
  <Ad>
    <Inline>
      ...
    <Creatives>
      <Creative id="1" sequence="1">
        <Linear skipoffset="00:00:05">
          <Duration>00:00:15</Duration>
          <MediaFiles>
            <MediaFile id="EMT" delivery="progressive" width="640" height="360"
              type="video/mp4" bitrate="143" scalable="true" maintainAspectRatio="true"><![
[CDATA[https://ads.com/file.mp4]]></MediaFile>
            </MediaFiles>
          </Linear>
        </Creative>
      </Creatives>
      ...
    </Inline>
  </Ad>
</VAST>
```



```

</Inline>
</Ad>
</VAST>

```

The following client-side tracking payload JSON response shows the ad metadata inside the ads array. The array contains the skipOffset value that MediaTailor obtained from the VAST response.

```

{
  "avails": [
    {
      "adBreakTrackingEvents": [],
      "adMarkerDuration": null,
      "ads": [
        {
          "adId": "1",
          "adParameters": "",
          "adProgramDateTime": "2023-07-31T16:11:40.693Z",
          "adSystem": "2.0",
          "adTitle": "AD-skiing-15",
          "adVerifications": [],
          "companionAds": [...],
          "creativeId": "1",
          "creativeSequence": "1",
          "duration": "PT15.015S",
          "durationInSeconds": 15.015,
          "extensions": [],
          "mediaFiles": {
            "mediaFilesList": [],
            "mezzanine": ""
          },
          "skipOffset": "00:00:05",
          "startTime": "PT9.943S",
          "startTimeInSeconds": 9.943,
          "trackingEvents": [
            {
              "beaconUrls": [
                "https://adserverbeaconing.com/v1/impression"
              ],
              "duration": "PT15.015S",
              "durationInSeconds": 15.015,
              "eventId": "2697726",
              "eventProgramDateTime": null,

```

```
        "eventType": "impression",
        "startTime": "PT9.943S",
        "startTimeInSeconds": 9.943
      }
    ],
    "vastAdId": ""
  }
],
"availId": "2697726",
"availProgramDateTime": "2023-07-31T16:11:40.693Z",
"duration": "PT15.015S",
"durationInSeconds": 15.015,
"meta": null,
"nonLinearAdsList": [],
"startTime": "PT9.943S",
"startTimeInSeconds": 9.943
}
],
"dashAvailabilityStartTime": null,
"hlsAnchorMediaSequenceNumber": null,
"nextToken": "",
"nonLinearAvails": []
}
```

Ad click-through

Click-through URIs allow advertisers to measure how successful an ad is in capturing viewers' attention. After a viewer clicks the active video frame of an ad in progress, a web browser opens the URI for the advertiser's home page or campaign landing page. The player developer determines the click behavior, such as overlaying a button or label on the ad video, with a message to click to learn more. Player developers often pause the ad's video after viewers click on the active video frame.



Click here for deals on Amazon.com

MediaTailor can parse and make available any linear video click-through event URLs returned in the VAST response. The following VAST response shows an ad click-through example.

```
<?xml version="1.0" encoding="UTF-8"?>
<VAST xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="vast.xsd" version="3.0">
  <Ad>
    <Inline>
      ...
    <Creatives>
      <Creative id="1" sequence="1">
        <Linear>
          <Duration>00:00:15</Duration>
          <MediaFiles>
```

```

        <MediaFile id="EMT" delivery="progressive" width="1280" height="720"
        type="video/mp4" bitrate="143" scalable="true" maintainAspectRatio="true"><![
[CDATA[https://ads.com/file.mp4]]></MediaFile>
        </MediaFiles>
        <VideoClicks>
            <ClickThrough id="EMT"><![CDATA[https://aws.amazon.com]]></ClickThrough>
            <ClickTracking id="EMT"><![CDATA[https://myads.com/beaconing/
event=clicktracking]]></ClickTracking>
        </VideoClicks>
    </Linear>
</Creative>
</Creatives>
...
</InLine>
</Ad>
</VAST>

```

The following client-side tracking payload JSON response shows how MediaTailor displays the click-through and click-tracking URLs inside the `trackingEvents` array. The `clickThrough` event type represents the click-through ad, and the `clickTracking` event type represents the click-tracking URL.

```

{
  "avails": [
    {
      "adBreakTrackingEvents": [],
      "adMarkerDuration": null,
      "ads": [
        {
          "adId": "1",
          "adParameters": "",
          "adProgramDateTime": "2023-07-31T16:53:40.577Z",
          "adSystem": "2.0",
          "adTitle": "1",
          "adVerifications": [],
          "companionAds": [],
          "creativeId": "000006",
          "creativeSequence": "1",
          "duration": "PT14.982S",
          "durationInSeconds": 14.982,
          "extensions": [],
          "mediaFiles": {
            "mediaFilesList": [],

```

```
    "mezzanine": ""
  },
  "skipOffset": null,
  "startTime": "PT39.339S",
  "startTimeInSeconds": 39.339,
  "trackingEvents": [
    {
      "beaconUrls": [
        "https://myads.com/beaconing/event=impression"
      ],
      "duration": "PT14.982S",
      "durationInSeconds": 14.982,
      "eventId": "2698188",
      "eventProgramDateTime": null,
      "eventType": "impression",
      "startTime": "PT39.339S",
      "startTimeInSeconds": 39.339
    },
    {
      "beaconUrls": [
        "https://aws.amazon.com"
      ],
      "duration": "PT14.982S",
      "durationInSeconds": 14.982,
      "eventId": "2698188",
      "eventProgramDateTime": null,
      "eventType": "clickThrough",
      "startTime": "PT39.339S",
      "startTimeInSeconds": 39.339
    },
    {
      "beaconUrls": [
        "https://myads.com/beaconing/event=clicktracking"
      ],
      "duration": "PT14.982S",
      "durationInSeconds": 14.982,
      "eventId": "2698795",
      "eventProgramDateTime": null,
      "eventType": "clickTracking",
      "startTime": "PT39.339S",
      "startTimeInSeconds": 39.339
    }
  ],
  "vastAdId": ""
```

```

    }
  ],
  "availId": "2698188",
  "availProgramDateTime": "2023-07-31T16:53:40.577Z",
  "duration": "PT14.982S",
  "durationInSeconds": 14.982,
  "meta": null,
  "nonLinearAdsList": [],
  "startTime": "PT39.339S",
  "startTimeInSeconds": 39.339
}
],
"dashAvailabilityStartTime": null,
"hlsAnchorMediaSequenceNumber": null,
"nextToken": "UFQz0S4zMz1TXzIwMjMtMDctMzFUMTY6NTQ6MDQu0DA1Mzk2NTI5Wl8x",
"nonLinearAvails": []
}

```

Companion ads

A *companion ad* appears alongside a linear creative. Use companion ads to increase the effectiveness of an ad spot by displaying product, logo, and brand information. The display ad can feature Quick Response (QR) codes and clickable areas to promote audience engagement.

MediaTailor supports companion ads in the VAST response. It can pass through metadata from `StaticResource`, `iFrameResource`, and `HTMLResource` nodes, respectively.

The following VAST response shows an example location and format of the linear ad and companion ad.

```

<?xml version="1.0" encoding="UTF-8"?>
<VAST xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="vast.xsd" version="3.0">
  <Ad>
    <InLine>
      ...
    <Creatives>
      <Creative id="1" sequence="1">
        <Linear>
          <Duration>00:00:10</Duration>
          <MediaFiles>

```

```

        <MediaFile id="EMT" delivery="progressive" width="640" height="360"
        type="video/mp4" bitrate="143" scalable="true" maintainAspectRatio="true"><![
[CDATA[https://ads.com/file.mp4]]></MediaFile>
    </MediaFiles>
</Linear>
</Creative>
<Creative id="2" sequence="1">
    <CompanionAds>
        <Companion id="2" width="300" height="250">
            <StaticResource creativeType="image/png"><![CDATA[https://emt.com/companion/9973499273]]></StaticResource>
            <TrackingEvents>
                <Tracking event="creativeView"><![CDATA[https://beacon.com/1]]></
Tracking>
            </TrackingEvents>
            <CompanionClickThrough><![CDATA[https://beacon.com/2]]></
CompanionClickThrough>
        </Companion>
        <Companion id="3" width="728" height="90">
            <StaticResource creativeType="image/png"><![CDATA[https://emt.com/companion/1238901823]]></StaticResource>
            <TrackingEvents>
                <Tracking event="creativeView"><![CDATA[https://beacon.com/3]]></
Tracking>
            </TrackingEvents>
            <CompanionClickThrough><![CDATA[https://beacon.com/4]]></
CompanionClickThrough>
        </Companion>
    </CompanionAds>
</Creative>
</Creatives>
    ...
</InLine>
</Ad>
</VAST>

```

The data appears in the client-side tracking response in the `/avail/x/ads/y/companionAds` list. Each linear creative can contain up to 6 companion ads. As shown in the example below, the companion ads appear in a list

Note

As a best practice, application developers should implement logic to explicitly remove or unload the companion ad at the end of the creative.

```
{
  "avails": [
    {
      "adBreakTrackingEvents": [],
      "adMarkerDuration": null,
      "ads": [
        {
          "adId": "0",
          "adParameters": "",
          "adProgramDateTime": null,
          "adSystem": "EMT",
          "adTitle": "sample",
          "adVerifications": [],
          "companionAds": [
            {
              "adParameters": null,
              "altText": null,
              "attributes": {
                "adSlotId": null,
                "apiFramework": null,
                "assetHeight": null,
                "assetWidth": null,
                "expandedHeight": null,
                "expandedWidth": null,
                "height": "250",
                "id": "2",
                "pxratio": null,
                "renderingMode": null,
                "width": "300"
              },
              "companionClickThrough": "https://beacon.com/2",
              "companionClickTracking": null,
              "htmlResource": null,
              "iFrameResource": null,
              "sequence": "1",
              "staticResource": "https://emt.com/companion/9973499273",
            }
          ]
        }
      ]
    }
  ]
}
```



```
    "trackingEvents": [
      {
        "beaconUrls": [
          "https://beacon.com/1"
        ],
        "eventType": "creativeView"
      }
    ],
  },
  {
    "adParameters": null,
    "altText": null,
    "attributes": {
      "adSlotId": null,
      "apiFramework": null,
      "assetHeight": null,
      "assetWidth": null,
      "expandedHeight": null,
      "expandedWidth": null,
      "height": "90",
      "id": "3",
      "pxratio": null,
      "renderingMode": null,
      "width": "728"
    },
    "companionClickThrough": "https://beacon.com/4",
    "companionClickTracking": null,
    "htmlResource": null,
    "iFrameResource": null,
    "sequence": "1",
    "staticResource": "https://emt.com/companion/1238901823",
    "trackingEvents": [
      {
        "beaconUrls": [
          "https://beacon.com/3"
        ],
        "eventType": "creativeView"
      }
    ]
  }
],
"creativeId": "1",
"creativeSequence": "1",
"duration": "PT10S",
```

```

        "durationInSeconds": 10,
        "extensions": [],
        "mediaFiles": {
            "mediaFilesList": [],
            "mezzanine": ""
        },
        "skipOffset": null,
        "startTime": "PT0S",
        "startTimeInSeconds": 0,
        "trackingEvents": [
            {
                "beaconUrls": [
                    "https://beacon.com/impression/1"
                ],
                "duration": "PT10S",
                "durationInSeconds": 10,
                "eventId": "0",
                "eventProgramDateTime": null,
                "eventType": "impression",
                "startTime": "PT0S",
                "startTimeInSeconds": 0
            }
        ],
        "vastAdId": ""
    }
],
"availId": "0",
"availProgramDateTime": null,
"duration": "PT10S",
"durationInSeconds": 10,
"meta": null,
"nonLinearAdsList": [],
"startTime": "PT0S",
"startTimeInSeconds": 0
}
],
"dashAvailabilityStartTime": null,
"hlsAnchorMediaSequenceNumber": null,
"nextToken": "UFQxMFNFmJyAyMy0wNy0wNlQyMToxMDowOC42NzQ4NDA1NjJaXzE%3D",
"nonLinearAvails": []
}

```

Interactive ads (SIMID)

SecureInteractive Media Interface Definition (SIMID) is a standard for interactive advertising that was introduced in the VAST 4.x standard from the Interactive Advertising Bureau (IAB). SIMID decouples the loading of interactive elements from the primary linear creative on the player, referencing both in the VAST response. MediaTailor stitches in the primary creative to maintain the playback experience, and places metadata for the interactive components in the client-side tracking response.

In the following example VAST 4 response, the SIMID payload is inside the `InteractiveCreativeFile` node.

```
<?xml version="1.0"?>
<VAST xmlns:xsi="https://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="vast.xsd" version="3.0">
  <Ad id="1234567">
    <Inline>
      <AdSystem>SampleAdSystem</AdSystem>
      <AdTitle>Linear SIMID Example</AdTitle>
      <Description>SIMID example</Description>
      <Error>https://www.beacons.com/error</Error>
      <Impression>https://www.beacons.com/impression</Impression>
      <Creatives>
        <Creative sequence="1">
          <Linear>
            <Duration>00:00:15</Duration>
            <TrackingEvents>
              ...
            </TrackingEvents>
            <VideoClicks>
              <ClickThrough id="123">https://aws.amazon.com</ClickThrough>
              <ClickTracking id="123">https://www.beacons.com/click</ClickTracking>
            </VideoClicks>
            <MediaFiles>
              <MediaFile delivery="progressive" type="video/mp4">
https://interactive-ads.com/interactive-media-ad-sample/media/file.mp4
              </MediaFile>
              <InteractiveCreativeFile type="text/html" apiFramework="SIMID"
                variableDuration="true">
https://interactive-ads.com/interactive-media-ad-sample/sample\_simid.html
              </InteractiveCreativeFile>
```

```

        </MediaFiles>
    </Linear>
</Creative>
</Creatives>
</InLine>
</Ad>
</VAST>

```

In the following VAST 3 response, the SIMID payload is inside the Extensions node.

```

<?xml version="1.0"?>
<VAST xmlns:xsi="https://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="vast.xsd" version="3.0">
  <Ad id="1234567">
    <InLine>
      <AdSystem>SampleAdSystem</AdSystem>
      <AdTitle>Linear SIMID Example</AdTitle>
      <Description>SIMID example</Description>
      <Impression>https://www.beacons.com/impression</Impression>
      <Creatives>
        <Creative id="1" sequence="1">
          <Linear>
            <Duration>00:00:15</Duration>
            <TrackingEvents>
              ...
            </TrackingEvents>
            <VideoClicks>
              <ClickThrough id="123">https://aws.amazon.com</ClickThrough>
              <ClickTracking id="123">https://myads.com/beaconing/event=clicktracking</
ClickTracking>
            </VideoClicks>
            <MediaFiles>
              <MediaFile delivery="progressive" type="video/mp4">
https://interactive-ads.com/interactive-media-ad-
sample/media/file.mp4
              </MediaFile>
            </MediaFiles>
          </Linear>
        </Creative>
      </Creatives>
      <Extensions>
        <Extension type="InteractiveCreativeFile">

```

```

    <InteractiveCreativeFile type="text/html" apiFramework="SIMID"
variableDuration="true">
        https://interactive-ads.com/interactive-media-ad-sample/sample_simid.html
    </InteractiveCreativeFile>
</Extension>
</Extensions>
</InLine>
</Ad>
</VAST>

```

In the following client-side tracking response, the SIMID data appears in the `/avails/x/ads/y/extensions` list.

```

{
  "avails": [
    {
      "adBreakTrackingEvents": [],
      "adMarkerDuration": null,
      "ads": [
        {
          "adId": "1",
          "adParameters": "",
          "adProgramDateTime": "2023-07-31T16:53:40.577Z",
          "adSystem": "2.0",
          "adTitle": "Linear SIMID Example",
          "adVerifications": [],
          "companionAds": [],
          "creativeId": "1",
          "creativeSequence": "1",
          "duration": "PT14.982S",
          "durationInSeconds": 14.982,
          "extensions": [
            {
              "content": "<InteractiveCreativeFile type=\"text/html\" apiFramework=
\"SIMID\" variableDuration=\"true\">\nhttps://interactive-ads.com/interactive-media-ad-
sample/sample_simid.html</InteractiveCreativeFile>",
              "type": "InteractiveCreativeFile"
            }
          ],
          "mediaFiles": {
            "mediaFilesList": [],
            "mezzanine": ""
          }
        }
      ]
    }
  ]
}

```

```
"skipOffset": null,
"startTime": "PT39.339S",
"startTimeInSeconds": 39.339,
"trackingEvents": [
  {
    "beaconUrls": [
      "https://myads.com/beaconing/event=impression"
    ],
    "duration": "PT14.982S",
    "durationInSeconds": 14.982,
    "eventId": "2698188",
    "eventProgramDateTime": null,
    "eventType": "impression",
    "startTime": "PT39.339S",
    "startTimeInSeconds": 39.339
  },
  {
    "beaconUrls": [
      "https://aws.amazon.com"
    ],
    "duration": "PT14.982S",
    "durationInSeconds": 14.982,
    "eventId": "2698188",
    "eventProgramDateTime": null,
    "eventType": "clickThrough",
    "startTime": "PT39.339S",
    "startTimeInSeconds": 39.339
  },
  {
    "beaconUrls": [
      "https://myads.com/beaconing/event=clicktracking"
    ],
    "duration": "PT14.982S",
    "durationInSeconds": 14.982,
    "eventId": "2698795",
    "eventProgramDateTime": null,
    "eventType": "clickTracking",
    "startTime": "PT39.339S",
    "startTimeInSeconds": 39.339
  }
],
"vastAdId": ""
}
```

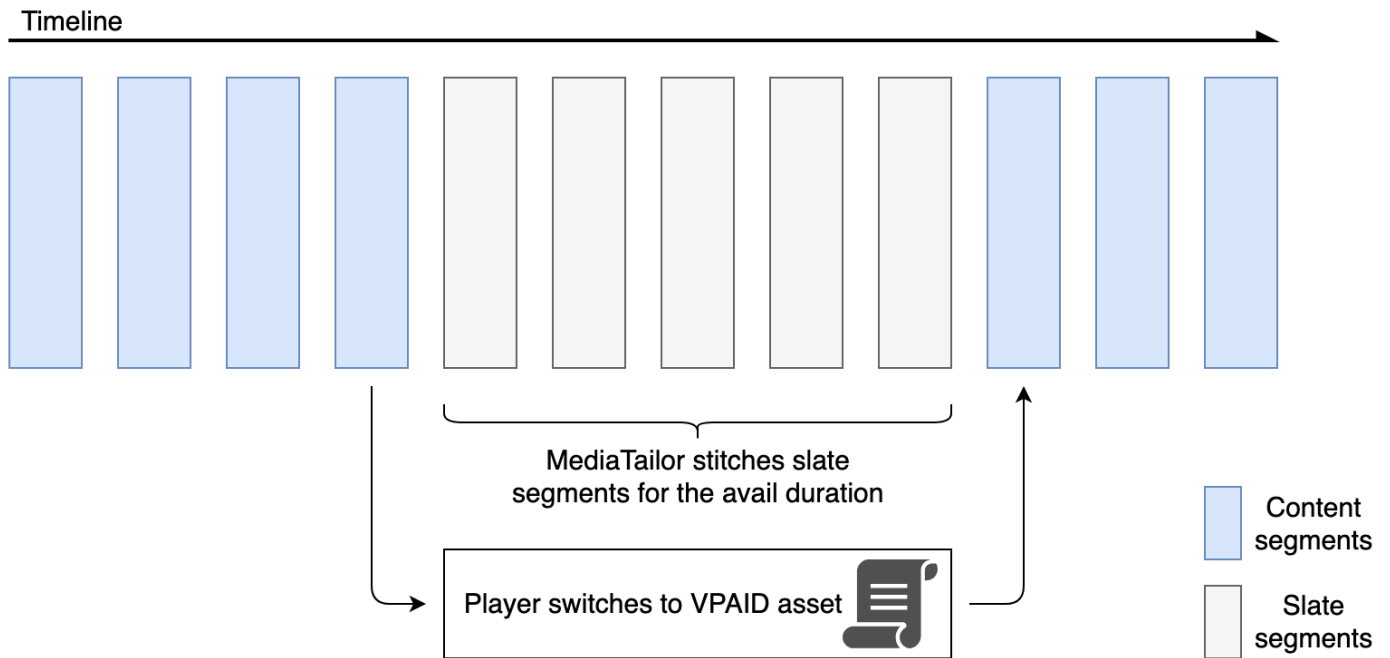
```
    "availId": "2698188",
    "availProgramDateTime": "2023-07-31T16:53:40.577Z",
    "duration": "PT14.982S",
    "durationInSeconds": 14.982,
    "meta": null,
    "nonLinearAdsList": [],
    "startTime": "PT39.339S",
    "startTimeInSeconds": 39.339
  }
],
"dashAvailabilityStartTime": null,
"hlsAnchorMediaSequenceNumber": null,
"nextToken": "UFQz0S4zMz1TXzIwMjMtMDctMzFUMTY6NTQ6MDQu0DA1Mzk2NTI5Wl8x",
"nonLinearAvails": []
}
```

Interactive ads (VPAID)

The *Video Player Ad Interface Definition* (VPAID) specifies the protocol between the ad and the video player that enables ad interactivity and other functionality. For live streams, MediaTailor supports the VPAID format by stitching slate segments in for the duration of the avail, and placing metadata for the VPAID creatives in the client-side tracking response that the video player consumes. The player downloads the VPAID files and plays the linear creative and executes the client's scripts. The player should *not* ever play the slate segments.

Note

VPAID is deprecated as of VAST 4.1.



The following example shows the VPAID content in the VAST response.

```
<?xml version="1.0"?>
<VAST xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="vast.xsd" version="3.0">
  <Ad id="1234567">
    <Inline>
      <AdSystem>GDFP</AdSystem>
      <AdTitle>VPAID</AdTitle>
      <Description>Vpaid Linear Video Ad</Description>
      <Error>http://www.example.com/error</Error>
      <Impression>http://www.example.com/impression</Impression>
      <Creatives>
        <Creative sequence="1">
          <Linear>
            <Duration>00:00:00</Duration>
            <TrackingEvents>
              <Tracking event="start">http://www.example.com/start</Tracking>
              <Tracking event="firstQuartile">http://www.example.com/firstQuartile</
Tracking>
              <Tracking event="midpoint">http://www.example.com/midpoint</Tracking>
              <Tracking event="thirdQuartile">http://www.example.com/thirdQuartile</
Tracking>
```



```

    <Tracking event="complete">http://www.example.com/complete</Tracking>
    <Tracking event="mute">http://www.example.com/mute</Tracking>
    <Tracking event="unmute">http://www.example.com/unmute</Tracking>
    <Tracking event="rewind">http://www.example.com/rewind</Tracking>
    <Tracking event="pause">http://www.example.com/pause</Tracking>
    <Tracking event="resume">http://www.example.com/resume</Tracking>
    <Tracking event="fullscreen">http://www.example.com/fullscreen</Tracking>
    <Tracking event="creativeView">http://www.example.com/creativeView</
Tracking>
    <Tracking event="acceptInvitation">http://www.example.com/
acceptInvitation</Tracking>
  </TrackingEvents>
  <AdParameters><![CDATA[ {"videos":[ {"url":"https://my-ads.com/interactive-
media-ads/media/media_linear_VPAID.mp4", "mimetype":"video/mp4"}]} ]]></AdParameters>
  <VideoClicks>
    <ClickThrough id="123">http://google.com</ClickThrough>
    <ClickTracking id="123">http://www.example.com/click</ClickTracking>
  </VideoClicks>
  <MediaFiles>
    <MediaFile delivery="progressive" apiFramework="VPAID" type="application/
javascript" width="640" height="480"> https://googleads.github.io/googleads-ima-html5/
vpaid/linear/VpaidVideoAd.js </MediaFile>
  </MediaFiles>
</Linear>
</Creative>
</Creatives>
</InLine>
</Ad>
</VAST>

```

The following example shows the tracking information.

```

{
  "avails": [
    {
      "adBreakTrackingEvents": [],
      "adMarkerDuration": null,
      "ads": [
        {
          "adId": "1",
          "adParameters": "",
          "adProgramDateTime": "2023-07-31T16:53:40.577Z",
          "adSystem": "2.0",

```

```
"adTitle": "1",
"adVerifications": [],
"companionAds": [],
"creativeId": "00006",
"creativeSequence": "1",
"duration": "PT14.982S",
"durationInSeconds": 14.982,
"extensions": [],
"mediaFiles": {
  "mediaFilesList": [],
  "mezzanine": ""
},
"skipOffset": null,
"startTime": "PT39.339S",
"startTimeInSeconds": 39.339,
"trackingEvents": [
  {
    "beaconUrls": [
      "https://myads.com/beaconing/event=impression"
    ],
    "duration": "PT14.982S",
    "durationInSeconds": 14.982,
    "eventId": "2698188",
    "eventProgramDateTime": null,
    "eventType": "impression",
    "startTime": "PT39.339S",
    "startTimeInSeconds": 39.339
  },
  {
    "beaconUrls": [
      "https://aws.amazon.com"
    ],
    "duration": "PT14.982S",
    "durationInSeconds": 14.982,
    "eventId": "2698188",
    "eventProgramDateTime": null,
    "eventType": "clickThrough",
    "startTime": "PT39.339S",
    "startTimeInSeconds": 39.339
  },
  {
    "beaconUrls": [
      "https://myads.com/beaconing/event=clicktracking"
    ],
```

```

        "duration": "PT14.982S",
        "durationInSeconds": 14.982,
        "eventId": "2698795",
        "eventProgramDateTime": null,
        "eventType": "clickTracking",
        "startTime": "PT39.339S",
        "startTimeInSeconds": 39.339
    }
],
    "vastAdId": ""
}
],
    "availId": "2698188",
    "availProgramDateTime": "2023-07-31T16:53:40.577Z",
    "duration": "PT14.982S",
    "durationInSeconds": 14.982,
    "meta": null,
    "nonLinearAdsList": [],
    "startTime": "PT39.339S",
    "startTimeInSeconds": 39.339
}
],
    "dashAvailabilityStartTime": null,
    "hlsAnchorMediaSequenceNumber": null,
    "nextToken": "UFQz0S4zMzlTXzIwMjMtMDctMzFUMTY6NTQ6MDQu0DA1Mzk2NTI5Wl8x",
    "nonLinearAvails": []
}
{
    "avails": [
        {
            "adBreakTrackingEvents": [],
            "adMarkerDuration": null,
            "ads": [
                {
                    "adId": "2922274",
                    "adParameters": "",
                    "adProgramDateTime": "2023-08-14T19:49:53.998Z",
                    "adSystem": "Innovid Ads",
                    "adTitle": "VPAID",
                    "adVerifications": [],
                    "companionAds": [],
                    "creativeId": "",
                    "creativeSequence": "",
                    "duration": "PT16.016S",
                    "durationInSeconds": 16.016,

```

```

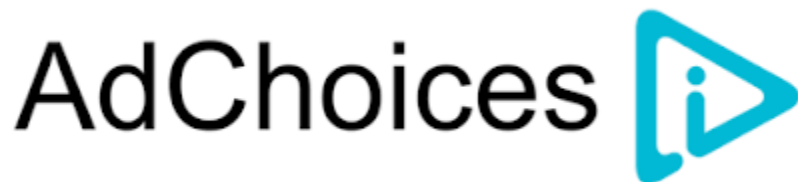
    "extensions": [],
    "mediaFiles": {
      "mediaFilesList": [
        {
          "apiFramework": "VPAID",
          "bitrate": 0,
          "codec": null,
          "delivery": "progressive",
          "height": 9,
          "id": "",
          "maintainAspectRatio": false,
          "maxBitrate": 0,
          "mediaFileUri": "http://my-ads.com/mobileapps/js/vpaid/1h41kg?
cb=178344c0-8e67-281a-58ca-962e4987cd60&deviceid=&ivc=",
          "mediaType": "application/javascript",
          "minBitrate": 0,
          "scalable": false,
          "width": 16
        }
      ],
      "mezzanine": "http://my-ads.com/mobileapps/js/vpaid/1h41kg?
cb=178344c0-8e67-281a-58ca-962e4987cd60&deviceid=&ivc="
    },
    "skipOffset": null,
    "startTime": "PT8M42.289S",
    "startTimeInSeconds": 522.289,
    "trackingEvents": [
      {
        "beaconUrls": [
          "about:blank"
        ],
        "duration": "PT16.016S",
        "durationInSeconds": 16.016,
        "eventId": "2922274",
        "eventProgramDateTime": null,
        "eventType": "impression",
        "startTime": "PT8M42.289S",
        "startTimeInSeconds": 522.289
      }
    ],
    "vastAdId": "1h41kg"
  }
],
  "availId": "2922274",

```

```
    "availProgramDateTime": "2023-08-14T19:49:53.998Z",
    "duration": "PT16.016S",
    "durationInSeconds": 16.016,
    "meta": null,
    "nonLinearAdsList": [],
    "startTime": "PT8M42.289S",
    "startTimeInSeconds": 522.289
  }
],
"dashAvailabilityStartTime": null,
"hlsAnchorMediaSequenceNumber": null,
"nextToken": "UFQ4TTQyLjI4OVNfMjAyMy0wOC0xNFQxOT01MDo0MS4zOTc5MjAzODVaXzE%3D",
"nonLinearAvails": []
}
```

Icons for Google Why This Ad (WTA)

AdChoices is an industry standard that provides viewers with information about the ads they see, including how those ads were targeted to them.



The MediaTailor client-side tracking API supports icon metadata carried in the VAST extensions node of the VAST response. For more information about WTA in the VAST response, see [this sample VAST XML response](#).

Note

MediaTailor currently supports VAST version 3 only.

```
<VAST>
  <Ad>
    <InLine>
      ...
    <Extensions>
      <Extension type="IconClickFallbackImages">
```

```

    <IconClickFallbackImages program="GoogleWhyThisAd">
      <IconClickFallbackImage width="400" height="150">
        <AltText>Alt icon fallback</AltText>
        <StaticResource creativeType="image/png"><![CDATA[https://
storage.googleapis.com/interactive-media-ads/images/wta_dialog.png]]></StaticResource>
      </IconClickFallbackImage>
    </IconClickFallbackImages>
    <IconClickFallbackImages program="AdChoices">
      <IconClickFallbackImage width="400" height="150">
        <AltText>Alt icon fallback</AltText>
        <StaticResource creativeType="image/png"><![CDATA[https://
storage.googleapis.com/interactive-media-ads/images/wta_dialog.png?size=1x]]></
StaticResource>
      </IconClickFallbackImage>
      <IconClickFallbackImage width="800" height="300">
        <AltText>Alt icon fallback</AltText>
        <StaticResource creativeType="image/png"><![CDATA[https://
storage.googleapis.com/interactive-media-ads/images/wta_dialog.png?size=2x]]></
StaticResource>
      </IconClickFallbackImage>
    </IconClickFallbackImages>
  </Extension>
</Extensions>
</InLine>
</Ad>
</VAST>

```

The following example shows the client-side tracking response in the `/avails/x/ads/y/` extensions list.

```

{
  "avails": [
    {
      "adBreakTrackingEvents": [],
      "adMarkerDuration": null,
      "ads": [
        {
          "adId": "0",
          "adParameters": "",
          "adProgramDateTime": null,
          "adSystem": "GDFP",
          "adTitle": "Google Why This Ad VAST 3 Sample",
          "adVerifications": [],

```

```

    "companionAds": [],
    "creativeId": "7891011",
    "creativeSequence": "1",
    "duration": "PT10S",
    "durationInSeconds": 10,
    "extensions": [
      {
        "content": "<IconClickFallbackImages program=\"GoogleWhyThisAd\">
          <IconClickFallbackImage height=\"150\" width=\"400\">
            <AltText>Alt icon fallback</AltText>
            <StaticResource creativeType=\"image/png\"><![CDATA[https://
storage.googleapis.com/interactive-media-ads/images/wta_dialog.png]]>
          </StaticResource>
        </IconClickFallbackImage>
      </IconClickFallbackImages>
      <IconClickFallbackImages program=\"AdChoices\">
        <IconClickFallbackImage height=\"150\" width=\"400\">
          <AltText>Alt icon fallback</AltText>
          <StaticResource creativeType=\"image/png\"><![CDATA[https://
storage.googleapis.com/interactive-media-ads/images/wta_dialog.png?size=1x]]>
        </StaticResource>
      </IconClickFallbackImage>
      <IconClickFallbackImage height=\"300\" width=\"800\">
        <AltText>Alt icon fallback</AltText>
        <StaticResource creativeType=\"image/png\"><![CDATA[https://
storage.googleapis.com/interactive-media-ads/images/wta_dialog.png?size=2x]]>
        </StaticResource>
      </IconClickFallbackImage>
    </IconClickFallbackImages>",
        "type": "IconClickFallbackImages"
      }
    ],
    "mediaFiles": {
      "mediaFilesList": [],
      "mezzanine": ""
    },
    "skipOffset": "00:00:03",
    "startTime": "PT0S",
    "startTimeInSeconds": 0,
    "trackingEvents": [
      {
        "beaconUrls": [
          "https://example.com/view"
        ]
      }
    ],

```

```

        "duration": "PT10S",
        "durationInSeconds": 10,
        "eventId": "0",
        "eventProgramDateTime": null,
        "eventType": "impression",
        "startTime": "PT0S",
        "startTimeInSeconds": 0
      }
    ],
    "vastAdId": "123456"
  }
],
"availId": "0",
"availProgramDateTime": null,
"duration": "PT10S",
"durationInSeconds": 10,
"meta": null,
"nonLinearAdsList": [],
"startTime": "PT0S",
"startTimeInSeconds": 0
}
],
"dashAvailabilityStartTime": null,
"hlsAnchorMediaSequenceNumber": null,
"nextToken": "UFQxMFNfMjAyMy0wNy0wNlQyMDo0MT0xNy45NDE4MDM0NDhaXzE%3D",
"nonLinearAvails": []
}

```

Client-side beaconing

With the client-side tracking `startTimeInSeconds` element, you can use MediaTailor to support beacons timing.

The following JSON response shows the main beacon types: impressions, start, quartiles, and completion.

Note

The Interactive Advertising Bureau (IAB) Video Impression Measurement Guidelines state that an impression requires the ad content to load client-side and, at a minimum, begin time to render into the player. For more information, see [Digital Video Ad Serving Template \(VAST\)](#) on the IAB website.


```
{
  "avails": [
    {
      "ads": [
        {
          "adId": "8104385",
          "duration": "PT15.100000078S",
          "durationInSeconds": 15.1,
          "startTime": "PT17.817798612S",
          "startTimeInSeconds": 17.817,
          "trackingEvents": [
            {
              "beaconUrls": [
                "http://exampleleadserver.com/tracking?event=impression"
              ],
              "duration": "PT15.100000078S",
              "durationInSeconds": 15.1,
              "eventId": "8104385",
              "eventType": "impression",
              "startTime": "PT17.817798612S",
              "startTimeInSeconds": 17.817
            },
            {
              "beaconUrls": [
                "http://exampleleadserver.com/tracking?event=start"
              ],
              "duration": "PT0S",
              "durationInSeconds": 0.0,
              "eventId": "8104385",
              "eventType": "start",
              "startTime": "PT17.817798612S",
              "startTimeInSeconds": 17.817
            },
            {
              "beaconUrls": [
                "http://exampleleadserver.com/tracking?event=firstQuartile"
              ],
              "duration": "PT0S",
              "durationInSeconds": 0.0,
              "eventId": "8104386",
              "eventType": "firstQuartile",
              "startTime": "PT21.592798631S",
              "startTimeInSeconds": 21.592
            }
          ]
        }
      ]
    }
  ]
}
```

```

    },
    {
      "beaconUrls": [
        "http://exampleleadserver.com/tracking?event=midpoint"
      ],
      "duration": "PT0S",
      "durationInSeconds": 0.0,
      "eventId": "8104387",
      "eventType": "midpoint",
      "startTime": "PT25.367798651S",
      "startTimeInSeconds": 25.367
    },
    {
      "beaconUrls": [
        "http://exampleleadserver.com/tracking?event=thirdQuartile"
      ],
      "duration": "PT0S",
      "durationInSeconds": 0.0,
      "eventId": "8104388",
      "eventType": "thirdQuartile",
      "startTime": "PT29.14279867S",
      "startTimeInSeconds": 29.142
    },
    {
      "beaconUrls": [
        "http://exampleleadserver.com/tracking?event=complete"
      ],
      "duration": "PT0S",
      "durationInSeconds": 0.0,
      "eventId": "8104390",
      "eventType": "complete",
      "startTime": "PT32.91779869S",
      "startTimeInSeconds": 32.917
    }
  ]
}
],
"availId": "8104385",
"duration": "PT15.100000078S",
"durationInSeconds": 15.1,
"startTime": "PT17.817798612S",
"startTimeInSeconds": 17.817
}
]

```

```
}
```

Hybrid mode with server-side ad beacons

MediaTailor supports a hybrid mode for session tracking. In this mode, the service emits playback-related ad-tracking events, but makes the complete client-side tracking payload available for the session.

To enable hybrid tracking using playback prefixes, from the player initialize a new MediaTailor playback session using a request in one of the following formats, according to your protocol:

Example: HLS format

```
POST master.m3u8
{
  "adsParams": {
    "deviceType": "ipad"
  },
  "reportingMode": "server"
}
```

Example: DASH format

```
POST manifest.mpd
{
  "adsParams": {
    "deviceType": "ipad"
  },
  "reportingMode": "server"
}
```

MediaTailor maintains the following tracking events in hybrid mode:

- Impression
- Start
- First quartile
- Midpoint
- Third quartile
- Complete
- breakStart (vmap)

- `breakEnd (vmap)`

Client-side ad-tracking integrations

This section describes integrations between MediaTailor and various client-side ad-tracking servers.

Topics

- [Open Measurement SDK](#)
- [Datazoom free player SDKs](#)
- [Roku Advertising Framework \(RAF\)](#)
- [TheoPlayer](#)
- [MediaTailor SDK](#)

Open Measurement SDK

The Interactive Advertising Bureau (IAB) Open Measurement SDK (OM SDK) facilitates third-party viewability and verification measurement for ads served to web-video and native-app environments.

For older VAST version 3 documents, verification code should be loaded with the Extension node, with extension type `AdVerifications`. The root of the extension node is an `AdVerifications` node with the same schema as the VAST 4.1 element.

To facilitate easier adoption of the OM SDK, MediaTailor has partnered with Datazoom to provide free player SDKs that are configured and verified for Open Measurement. For more information, see [Datazoom free player SDKs](#).

Note

MediaTailor currently supports VAST version 3 only.

Example: Verification node in VAST 3, prior to Version 4.1

```
...
<Extensions>
  <Extension type="AdVerifications">
    <AdVerifications>
```

```

    <Verification vendor="company.com-omid">
      <JavaScriptResource apiFramework="omid" browserOptional="true">
        <![CDATA[https://verification.com/omid_verification.js]]>
      </JavaScriptResource>
      <TrackingEvents>
        <Tracking event="verificationNotExecuted">
          <![CDATA[https://verification.com/trackingurl]]>
        </Tracking>
      </TrackingEvents>
      <VerificationParameters>
        <![CDATA[verification params key/value pairs]]>
      </VerificationParameters>
    </Verification>
  </AdVerifications>
</Extension>
</Extensions>

```

MediaTailor extracts the AdVerifications data from the <Extensions> node and places it into the adVerifications array in the client-side tracking response.

Example: adVerifications array in client-side tracking response

```

{
  "avails": [
    {
      "adBreakTrackingEvents": [],
      "adMarkerDuration": null,
      "ads": [
        {
          "adId": "3062770",
          "adParameters": "",
          "adProgramDateTime": "2023-08-23T16:25:40.914Z",
          "adSystem": "2.0",
          "adTitle": "AD-polarbear-15",
          "adVerifications": [
            {
              "executableResource": [],
              "javaScriptResource": [
                {
                  "apiFramework": "omid",
                  "browserOptional": "true",
                  "uri": "https://verification.com/omid_verification.js"
                }
              ]
            }
          ]
        }
      ]
    }
  ]
}

```

[illegible]

```

        "eventProgramDateTime": null,
        "eventType": "impression",
        "startTime": "PT10.11S",
        "startTimeInSeconds": 10.11
    }
],
    "vastAdId": ""
}
],
    "availId": "3062770",
    "availProgramDateTime": "2023-08-23T16:25:40.914Z",
    "duration": "PT14.982S",
    "durationInSeconds": 14.982,
    "meta": null,
    "nonLinearAdsList": [],
    "startTime": "PT10.11S",
    "startTimeInSeconds": 10.11
}
],
    "dashAvailabilityStartTime": null,
    "hlsAnchorMediaSequenceNumber": null,
    "nextToken": "UFQxMC4xMVNfMjAyMy0wOC0yM1QxNjoyNjoyNC4yNDYxMDIxOTBaXzE%3D",
    "nonLinearAvails": []
}

```

Note

Engage with the IAB Tech Lab to ensure that applications are certified annually to ensure compliance.

For more information about the OM SDK, see [Open Measurement SDK](#) on the IAB Tech Lab website.

Datazoom free player SDKs

To facilitate easier adoption of the player SDKs, MediaTailor has partnered with Datazoom to provide free player SDKs that are configured and tested with the [Client-side AWS Elemental MediaTailor integration with Google Ad Manager](#) and the IAB Tech [Open Measurement SDK](#).

The Datazoom player SDK supports these features:

- Live and VOD playlists
- DASH and HLS specifications
- Player vendor support for Bitmovin, exoplayer, Android media player, Apple AVPlayer, Brightcove, Chromecast Receiver, Dash.js, hls.js, JWPlayer, Shaka player, THEO player, Video.js, Roku and more
- IAB Tech Lab Open Measurement certification, where available on selected devices
- Click-through event handling
- Ad-event dispatchers, such as ad count down timers, ad overlay and non-linear events, ad-break start, ad-break end
- Client-side ad beaconing
- Google Programmatic Access Library (PAL) SDK, as an optional configuration setting

Datazoom also offers a paid analytics and telemetry service that the player SDKs support. Customers can opt into and control player SDK telemetry from the Datazoom management console. To access the Datazoom player SDKs and to learn more about the value-added telemetry and analytics service, use the contact information on the [Datazoom site](#).

Roku Advertising Framework (RAF)

The Roku Ad Framework (RAF) maintains a consistent ad experience across the Roku platform. All channels, including video advertisements, must meet Roku's certification requirements for RAF. Notably, the app must always use client-side event firing through RAF. MediaTailor, as a server-side ad insertion (SSAI) provider, supports client-side event firing. The RAFX SSAI Adapters provide interfaces to both SSAI manifest servers, or stitchers, and RAF. These interfaces include:

- Parsing the masterURL response and extracting playURL, AdURL, and ad metadata.
- Transforming MediaTailor SSAI ad metadata into RAF-usable ad metadata, and configuring RAF for playback.
- Observing stream events and timed metadata.
- Matching the stream events, ad metadata, and firing-event pixels on time.
- Pinging/polling the AdURL, as required by the MediaTailor SSAI manifest server, then parsing and re-configuring RAF.

For more information about SSAI adapters for RAF, see [Implementing Server-Side Ad Insertion Using Roku Adapters](#) on the Roku website.

TheoPlayer

TheoPlayer integration with MediaTailor does the following:

- Provides functionality to support MediaTailor client-side event tracking for HLS and DASH for both VOD and live workflows.
- Supports sending tracking beacons only for linear ads.
- Disables seeking during an ad. However, no logic is in place for playing an ad when the user seeks past the ad break.

For more information about SSAI in TheoPlayer, and to review the web, Android, iOS, and tvOS SDKs for MediaTailor, see [MediaTailor](#) on the TheoPlayer website.

MediaTailor SDK

AWS Elemental maintains a JavaScript-based software-development kit (SDK). AWS Elemental provides the SDK as-is, with no implied warranty. Use the SDK as a reference demonstration to streamline your onboarding to using MediaTailor. The SDK shows how to interact with the MediaTailor client-side tracking API. The SDK implements client-side ad tracking and reporting for HTML5-based players. The SDK initializes a MediaTailor client-side reporting session, then periodically requests ad-tracking information. During playback, the SDK emits ad tracking events when new ad events are detected.

The MediaTailor SDK supports these features:

- Live and VOD playlists
- DASH and HLS specifications
- Click-through event handling
- Ad-event dispatchers
- Custom event hooks
- Client-side ad beaconing. For more information about sending ad beacons, see [Client-side beaconing](#).

Note

Submit an AWS Support ticket to receive a sample JavaScript SDK for MediaTailor. You'll receive a download link for the package and its files.

Paging through ad beacons with GetTracking

Use the `GetTracking` endpoint to narrow the number of ad returned to a player. For instance, if a manifest window is wide, spanning a lot of time, the number of returned ad beacons can impact player performance.

`GetTracking` returns a `NextToken` value you can use to narrow the number of returned beacons by paging through the list of returned beacons. You can cycle through the `NextToken` values to find the desired value of an ad beacon's `StartTimeInSeconds` field.

- On the first call to `GetTracking`, all possible ads falling in the manifest window are returned, including a `NextToken` and value of each.
- If a `GetTracking` request does *not* include a `NextToken`, all ads in the manifest window are returned.
- If a `GetTracking` request contains a `NextToken` but there are no new beacons to return, MediaTailor returns the same value for `NextToken` that you sent on the original request.
- When there are no more beacons corresponding to an ad, `GetTracking` removes the ad from its response.
- Tokens from `GetTracking` expire after 24 hours. If a `NextToken` value is greater than 24 hours old, the next call to `GetTracking` returns a null-value `NextToken`.

Generalized calling sequence of GetTracking from player

From the client player, a `GetTracking` request is a POST with a request body that contains the `NextToken` and ads and beacons related to the token.

```
https://YouMediaTailorUrl/v1/tracking
{
    "NextToken": "value"
}
```

$$\left. \begin{array}{l} \cdot \\ \cdot \end{array} \right\}$$

The general sequence for using `GetTracking` with `NextToken` is as follows:

- ## 1. Make the first call to GetTracking.

All ads and beacons and the first NextToken for subsequent calls are returned.

2. If the value of NextToken is null, MediaTailor returns all ad beacons.
3. If the NextToken is expired, MediaTailor returns an HTTP return code 400 error message.

Make a fresh call to `GetTracking` to retrieve valid `NextTokens`.

4. Scan the entire response to find the `StartTimeInSeconds` of an ad beacon that is in the desired range.
5. Make a new call to `GetTracking` with the value of `NextToken` associated with the desired `StartTimeInSeconds`.
6. If necessary, cycle again through the returned ads until you find the exact ones you want to play.

Extended example

This example shows how to use GetTracking's NextToken to constrain the number of ad beacons returned to a player.

MediaTailor receives a `GetTracking` request. The response contains an ad with ID 9935407 and two beacons with `StartTimeInSeconds` values 52.286 and 48.332 seconds.

MediaTailor sends the JSON response with NextToken as follows:

```
{
  "NextToken": "JF57ITe48t1441mv7TmLKuZLroxDzfIslp6BiSNL1IImzPVMdN0lqrBYycgMbKEb",
  "avails": [
    {
      "ads": [
        {
          "adId": "9935407",
          "adVerifications": [],
          "companionAds": [],

```

```

    "creativeId": "",
    "creativeSequence": "",
    "duration": "PT15S",
    "durationInSeconds": 15,
    "extensions": [],
    "mediaFiles": {
      "mediaFilesList": [],
      "mezzanine": ""
    },
    "startTime": "PT30S",
    "StartTimeInSeconds": 45,
    "trackingEvents": [
      {
        "beaconUrls": [
          "http://adserver.com/tracking?event=Impression "
        ],
        "duration": "PT0S",
        "durationInSeconds": 0,
        "eventId": "9935414",
        "eventType": "secondQuartile",
        "startTime": "PT52.286S",
        "StartTimeInSeconds": 52.286
      },
      {
        "beaconUrls": [
          "http://adserver.com/tracking?event=firstQuartile"
        ],
        "duration": "PT0S",
        "durationInSeconds": 0,
        "eventId": "9935412",
        "eventType": "firstQuartile",
        "startTime": "PT48.332S",
        "StartTimeInSeconds": 48.332
      }
    ],
    "vastAdId": ""
  }
],
"startTime": "PT46.47S",
"StartTimeInSeconds": 46.47
}
]
}

```

On the next `GetTracking` request, MediaTailor responds with the `NextToken` value, `JF57ITe48t1441mv7TmLKuZLroxDzflslp6BiSNL1IJmzPVMDN0lqrBYycgMbKEb`.

MediaTailor responds with ads and beacons that match the `StartTimeInSeconds` that are set in the `NextToken` of the previous call.

Assume that now the response includes another ad with ID 9235407 in addition to the previous ad with ID 9935407. The beacons of ad ID 9235407 have `StartTimeInSecondss` 132.41 and 70.339.

MediaTailor iterates over all the beacons in the session to select the ones with `StartTimeInSeconds` greater than 52.286 seconds, which are beacon 3 and beacon 4 from the ad with ID 9235407:

```
{
  "NextToken": "ZkfknvbfsgdfbsDFRdffg12EdffecFRvhjyjfhdhfnjtsG5SDGN
  "avails": [
    {
      "ads": [
        {
          "adId": "9235407",
          "adVerifications": [],
          "companionAds": [],
          "creativeId": "",
          "creativeSequence": "",
          "duration": "PT15.816S",
          "durationInSeconds": 19.716,
          "extensions": [],
          "mediaFiles": {
            "mediaFilesList": [],
            "mezzanine": ""
          },
          "startTime": "PT2M0S",
          "StartTimeInSeconds": 120.0,
          "trackingEvents": [
            {
              "beaconUrls": [
                "http://adserver.com/tracking?event=complete"
              ],
              "duration": "PT0S",
```

```

        "durationInSeconds": 0,
        "eventId": "8935414",
        "eventType": "firstQuartile",
        "startTime": "PT1M10.330S",
        "StartTimeInSeconds": 70.339
    },
    {
        "beaconUrls": [
            "http://adserver.com/tracking?event=thirdQuartile"
        ],
        "duration": "PT0S",
        "durationInSeconds": 0,
        "eventId": "8935412",
        "eventType": "secondQuartile",
        "startTime": "PT2M12.41S",
        "StartTimeInSeconds": 132.41
    }
],
    "vastAdId": ""
},
],
    "startTime": "PT36.47S",
    "StartTimeInSeconds": 36.47
}
]
}

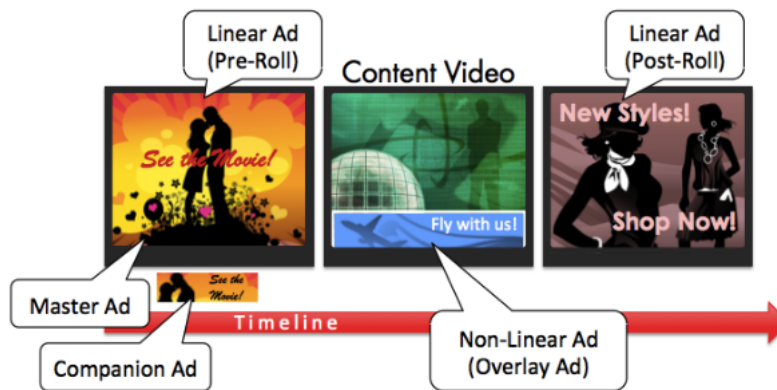
```

Overlay ads

For live-streaming workflows where you want to increase monetization without interrupting the viewing experience with mid-roll ads, you can leverage your current AWS Elemental MediaTailor integration to guide an advertising format that is rendered client-side. This type of advertising is known as *overlay ads*. Overlay ads are non-linear video ads that appear in the form of 'L-band ads,' 'non-linear video ads,' 'picture-in-picture ads,' 'motion overlays,' 'in-content advertising,' or 'frame ads.'

MediaTailor detects a SCTE-35 marker with segmentation type `id=0x38` as an in-band signal for an overlay ad-insertion opportunity. The SCTE-35 marker causes MediaTailor to send a request to the Ad Decision Server (ADS), which then responds with a non-linear ad payload in the VAST response. MediaTailor parses the VAST response in order to support overlay ad insertion.

MediaTailor does not perform any stitching of linear ads, but rather signals to the player that there's a non-linear overlay ad available to play. This signaling allows the player to fetch and correlate the non-linear ads to play from the client-side tracking endpoint. The player then handles the display, reporting, and other tasks related to those ads. For example, the player's developer can use a device SDK from a vendor that supports overlay-ad formats. For more information about client-side tracking integrations, see [Client-side ad-tracking integrations](#).



Topics

- [Prerequisites for using overlay ads with MediaTailor](#)
- [Getting started using overlay ads with MediaTailor](#)
- [Logging and metrics for overlay ads in MediaTailor](#)
- [Billing for overlay ads in MediaTailor](#)

Prerequisites for using overlay ads with MediaTailor

The following prerequisites apply when using overlay ads with MediaTailor:

- The workflow must be live, not video on demand (VOD).
- The Ad Decision Server (ADS) response must be configured to return only non-linear ads in the VAST response. MediaTailor ignores any linear ads for the purposes of ad stitching.
- The manifest must use a SCTE-35 time signal message with segmentation type `id=0x38` to invoke the overlay-ad feature.
- The streaming provider must have control of the client-device application and be integrated with the MediaTailor client-side tracking API.

Getting started using overlay ads with MediaTailor

This section explains how to get started using the overlay-ads feature of MediaTailor. You'll set up SCTE-35 signaling, configure Ad Decision Server (ADS) responses, and set up session-level control.

Topics

- [Enabling overlay ads](#)
- [Tracking overlay ads with client-side metadata](#)

Enabling overlay ads

MediaTailor support for overlay ads is enabled by default. A specific SCTE-35 ad-marker type in the manifest triggers the insertion of an overlay ad. Because some players might not support client-side rendering of overlay ads, you can disable the feature at the session level.

To disable overlay-ad support using HLS or DASH playback prefixes:

- From the player, initialize a new MediaTailor playback session using a request in one of the following formats, according to your protocol:
 - Example: HLS format

```
GET mediatailorURL/v1/master/hashed-account-id/origin-id/asset-id?  
aws.overlayAvails=off
```

- Example: DASH format

```
GET mediatailorURL/v1/master/hashed-account-id/origin-id/asset-id?  
aws.overlayAvails=off
```

To disable overlay-ad support using the session-initialization prefix:

- On the player, construct a JSON message body for the session initialization request to MediaTailor:
 - To disable ad-overlay support, add an `overlays` object as a top-level key with a value of `off`. The default `overlays` value is `on`.

- (Optional) Provide any parameters that MediaTailor then passes to the ADS inside an `adsParams` object. These parameters correspond to `[player_params.param]` settings in the ADS template URL of the MediaTailor configuration.

Example: HLS

```
POST master.m3u8
{
  "adsParams": {
    "deviceType": "ipad"
  },
  "overlayAvails": "off"
}
```

Example: DASH

```
POST manifest.mpd
{
  "adsParams": {
    "deviceType": "androidmobile"
  },
  "overlayAvails": "off"
}
```

Manifest signaling

MediaTailor trigger overlay-ads support when it sees a specific SCTE-35 marker in the manifest. The required signal is a splice command type 6, or time signal, that is a Provider Overlay Advertisement Start signal. This signal has a segmentation type id of `0x38`

The following example shows the `0x38` SCTE-35 marker in a JSON object.

```
{
  "tableId": 252,
  "selectionSyntaxIndicator": false,
  "privateIndicator": false,
  "sectionLength": 53,
  "protocolVersion": 0,
  "encryptedPacket": false,
  "encryptedAlgorithm": 0,
  "ptsAdjustment": 0,
```

```

"cwIndex": 0,
"tier": 4095,
"spliceCommandLength": 5,
"spliceCommandType": 6,
"spliceCommand": {
  "specified": true,
  "pts": 1800392
},
"descriptorLoopLength": 31,
"descriptors": [
  {
    "spliceDescriptorTag": 2,
    "descriptorLength": 29,
    "indentifier": "CUEI",
    "segmentationEventId": 158389361,
    "segmentationEventCancelIndicator": false,
    "programSegmentationFlag": true,
    "segmentationDurationFlag": true,
    "deliveryNotRestrictedFlag": false,
    "webDeliveryAllowedFlag": true,
    "noRegionalBlackoutFlag": true,
    "archiveAllowedFlag": true,
    "deviceResctrictions": 3,
    "segmentationDuration": 1350000,
    "segmentationUpidType": 9,
    "segmentationUpidLength": 7,
    "segmentationUpid": {
      "0": 111,
      "1": 118,
      "2": 101,
      "3": 114,
      "4": 108,
      "5": 97,
      "6": 121
    },
    "segmentationTypeId": 56,
    "segmentNum": 1,
    "segmentsExpected": 0
  }
],
"crc": 2510422713
}

```

The following example shows the SCTE-35 signal represented as a binary (base 32/hexadecimal) value:

```
0xfc303500000000000000fff00506fe001b78c8001f021d435545490970d4717fdf00000dbba009076f7665726c617
```

The following examples shows the SCTE-35 marker in both HLS and DASH manifests.

Example: HLS manifest

```
#EXTM3U
#EXT-X-VERSION:6
#EXT-X-TARGETDURATION:7
#EXT-X-MEDIA-SEQUENCE:419
#EXT-X-DISCONTINUITY-SEQUENCE:3
#EXT-X-PROGRAM-DATE-TIME:2023-08-15T04:30:09.231Z
#EXTINF:6.02,
https://aws.cloudfront.net/media/asset1/index1_00007.ts
#EXT-X-DISCONTINUITY
#EXT-X-KEY:METHOD=NONE
#EXT-X-PROGRAM-DATE-TIME:2023-08-15T04:30:15.251Z
#EXTINF:6.0,
https://aws.cloudfront.net/media/asset1/index1_00001.ts
#EXT-X-PROGRAM-DATE-TIME:2023-08-15T04:30:21.251Z
#EXTINF:4.0,
https://aws.cloudfront.net/media/asset1/index1_00002.ts
#EXT-X-DISCONTINUITY
#EXT-X-DATERANGE:ID="1692073825251-30-1",START-
DATE="2023-08-15T04:30:25.251Z",DURATION=10.0,PLANNED-DURATION=10.0,SCTE35-
OUT=0xfc30350000000000000000ff00506fe001b78c8001f021d435545490970d4717fdf00000dbba009076f7665726
#EXT-X-PROGRAM-DATE-TIME:2023-08-15T04:30:25.251Z
#EXTINF:2.0,
https://aws.cloudfront.net/media/asset1/index1_00003.ts
#EXT-X-PROGRAM-DATE-TIME:2023-08-15T04:30:27.251Z
#EXTINF:6.0,
https://aws.cloudfront.net/media/asset1/index1_00004.ts
#EXT-X-PROGRAM-DATE-TIME:2023-08-15T04:30:33.251Z
#EXTINF:2.0,
https://aws.cloudfront.net/media/asset1/index1_00005.ts
#EXT-X-DISCONTINUITY
#EXT-X-PROGRAM-DATE-TIME:2023-08-15T04:30:35.251Z
#EXTINF:4.0,
https://aws.cloudfront.net/media/asset1/index1_00006.ts
#EXT-X-PROGRAM-DATE-TIME:2023-08-15T04:30:39.251Z
```

```
#EXTINF:6.02,
https://aws.cloudfront.net/media/asset1/index1_00007.ts
```

Example: DASH manifest

```
<?xml version="1.0"?>
<MPD xmlns="urn:mpeg:dash:schema:mpd:2011" xmlns:scte35="urn:scte:scte35:2013:xml"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  availabilityStartTime="2023-08-15T16:34:05.911Z" minBufferTime="PT30S"
  minimumUpdatePeriod="PT2S" profiles="urn:mpeg:dash:profile:isoff-live:2011"
  publishTime="2023-08-15T16:34:17.950Z" suggestedPresentationDelay="PT20S"
  timeShiftBufferDepth="PT1M30S" type="dynamic"
  xsi:schemaLocation="urn:mpeg:dash:schema:mpd:2011 http://standards.iso.org/ittf/
PubliclyAvailableStandards/MPEG-DASH_schema_files/DASH-MPD.xsd">
  <Period xmlns="urn:mpeg:dash:schema:mpd:2011" id="1692117245944_1" start="PT0.033S">
    <BaseURL>https://aws.cloudfront.net/out/v1/abc/123/def/</BaseURL>
    <EventStream schemeIdUri="urn:scte:scte35:2013:xml" timescale="90000">
      <Event duration="900000">
        <scte35:SpliceInfoSection protocolVersion="0" ptsAdjustment="0" tier="4095">
          <scte35:TimeSignal>
            <scte35:SpliceTime ptsTime="0"/>
          </scte35:TimeSignal>
          <scte35:SegmentationDescriptor segmentNum="0" segmentationDuration="900000"
segmentationEventCancelIndicator="false" segmentationEventId="1"
segmentationTypeId="56" segmentsExpected="0" subSegmentNum="0"
subSegmentsExpected="0">
            <scte35:SegmentationUpid segmentationUpidFormat="hexBinary"
segmentationUpidType="14">63736f7665726c6179</scte35:SegmentationUpid>
          </scte35:SegmentationDescriptor>
        </scte35:SpliceInfoSection>
      </Event>
    </EventStream>
    <AdaptationSet bitstreamSwitching="true" mimeType="video/mp4"
segmentAlignment="true" startWithSAP="1" subsegmentAlignment="true"
subsegmentStartsWithSAP="1">
      <Representation bandwidth="3000000" codecs="avc1.4D4028" frameRate="30/1"
height="1080" id="1" width="1920">
        <SegmentTemplate initialization="../
cf684d31ec9e451ca98d2349989f6c0a/855c733eed20493ab3cc1100750bcf0b/
index_video_1_0_init.mp4" media="../
cf684d31ec9e451ca98d2349989f6c0a/855c733eed20493ab3cc1100750bcf0b/index_video_1_0_
$Number$.mp4" presentationTimeOffset="0" startNumber="1" timescale="30000">
          <SegmentTimeline>
```

```

        <S d="60000" r="6" t="1000"/>
        <S d="30000" t="421000"/>
    </SegmentTimeline>
</SegmentTemplate>
</Representation>
<Representation bandwidth="2499968" codecs="avc1.4D4028" frameRate="30/1"
height="1080" id="2" width="1920">
    <SegmentTemplate initialization="../
cf684d31ec9e451ca98d2349989f6c0a/855c733eed20493ab3cc1100750bcf0b/
index_video_2_0_init.mp4" media="../
cf684d31ec9e451ca98d2349989f6c0a/855c733eed20493ab3cc1100750bcf0b/index_video_2_0_
$Number$.mp4" presentationTimeOffset="0" startNumber="1" timescale="30000">
        <SegmentTimeline>
            <S d="60000" r="6" t="1000"/>
            <S d="30000" t="421000"/>
        </SegmentTimeline>
    </SegmentTemplate>
</Representation>
<Representation bandwidth="2200000" codecs="avc1.4D401F" frameRate="30/1"
height="720" id="3" width="1280">
    <SegmentTemplate initialization="../
cf684d31ec9e451ca98d2349989f6c0a/855c733eed20493ab3cc1100750bcf0b/
index_video_3_0_init.mp4" media="../
cf684d31ec9e451ca98d2349989f6c0a/855c733eed20493ab3cc1100750bcf0b/index_video_3_0_
$Number$.mp4" presentationTimeOffset="0" startNumber="1" timescale="30000">
        <SegmentTimeline>
            <S d="60000" r="6" t="1000"/>
            <S d="30000" t="421000"/>
        </SegmentTimeline>
    </SegmentTemplate>
</Representation>
</AdaptationSet>
<AdaptationSet lang="eng" mimeType="audio/mp4" segmentAlignment="0">
    <Label>Alternate Audio</Label>
    <Representation audioSamplingRate="48000" bandwidth="128000" codecs="mp4a.40.2"
id="9">
        <AudioChannelConfiguration
schemeIdUri="urn:mpeg:dash:23003:3:audio_channel_configuration:2011" value="2"/>
        <SegmentTemplate initialization="../
cf684d31ec9e451ca98d2349989f6c0a/855c733eed20493ab3cc1100750bcf0b/
index_audio_9_0_init.mp4" media="../
cf684d31ec9e451ca98d2349989f6c0a/855c733eed20493ab3cc1100750bcf0b/index_audio_9_0_
$Number$.mp4" presentationTimeOffset="0" startNumber="1" timescale="48000">
            <SegmentTimeline>

```

```

        <S d="98304" t="0"/>
        <S d="96256" t="98304"/>
        <S d="95232" t="194560"/>
        <S d="96256" r="2" t="289792"/>
        <S d="95232" t="578560"/>
        <S d="46080" t="673792"/>
    </SegmentTimeline>
</SegmentTemplate>
</Representation>
</AdaptationSet>
</Period>
</MPD>

```

Ad Decision Server (ADS) response

The ADS response must contain one valid tracking event. At minimum, the tracking event can be an Impression tracking event. The tracking event should contain at least one NonLinear ad. This ad is the overlay ad, taking the form of a static, HTML, or iFrame resource.

```
<vmap AdBreak breaktype="linear" breakId="csoverlay"
```

If the VAST response is a VMAP with breakType of nonlinear, the avail metadata is inside the nonLinearAvails root object. If the VAST response is a VMAP with a breakType of linear, or is a plain VAST response without VMAP, the avail metadata is inside the avails root object.

The following VAST response is a wrapped VMAP response with a breakType value of linear.

In addition to the wrapped VMAP response, MediaTailor also supports a wrapped VMAP response with a breakType value of nonlinear, and a plain VAST response.

```

<?xml version="1.0" encoding="utf-8"?>
<vmap:VMAP xmlns:vmap="http://www.iab.net/vmap-1.0" version="1.0">
  <vmap:AdBreak breakType="linear" breakId="csoverlay">
    <vmap:AdSource allowMultipleAds="true" followRedirects="true" id="1">
      <vmap:VASTAdData>
        <VAST xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="3.0"
xsi:noNamespaceSchemaLocation="vast.xsd">
          <Ad sequence="1">
            <Inline>
              <AdSystem>2.0</AdSystem>
              <AdTitle>2</AdTitle>

```

```

        <Impression><![CDATA[https://adserver.com/beacon=impression]]></
Impression>
        <Creatives>
            <Creative>
                <NonLinearAds>
                    <NonLinear width="640" height="360" id="18">
                        <StaticResource creativeType="text/js_ref"><![CDATA[https://
client-side-ads.com/tags/static/ctv-generic/overlay001.json?iv_geo_country%3DUS%26]]></
StaticResource>
                    </NonLinear>
                </NonLinearAds>
            </Creative>
        </Creatives>
    </InLine>
</Ad>
</VAST>
</vmap:VASTAdData>
</vmap:AdSource>
<vmap:TrackingEvents>
    <vmap:Tracking event="breakStart"><![CDATA[https://adserver.com/
beacon=breakstartimpression]]></vmap:Tracking>
    <vmap:Tracking event="breakEnd"><![CDATA[https://adserver.com/
beacon=breakendimpression]]></vmap:Tracking>
</vmap:TrackingEvents>
</vmap:AdBreak>
</vmap:VMAP>

```

Example 1: DASH manifest source to MediaTailor

```

<?xml version="1.0" encoding="utf-8"?>
<MPD xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="urn:mpeg:dash:schema:mpd:2011" xmlns:scte35="urn:scte:scte35:2013:xml"
  xsi:schemaLocation="urn:mpeg:dash:schema:mpd:2011 http://standards.iso.org/
  ittf/PubliclyAvailableStandards/MPEG-DASH_schema_files/DASH-MPD.xsd" id="201"
  type="dynamic" publishTime="2022-11-07T19:59:05+00:00" minimumUpdatePeriod="PT2S"
  availabilityStartTime="2022-11-07T06:57:11.250000+00:00" minBufferTime="PT10S"
  suggestedPresentationDelay="PT20.000S" timeShiftBufferDepth="PT58.999S"
  profiles="urn:mpeg:dash:profile:isoff-live:2011">
  <Period start="PT46827.601S" id="0" duration="PT88.321S">
    ...
  </Period>
  <Period start="PT46915.922S" id="45" duration="PT6.006S">
    <EventStream timescale="90000" schemeIdUri="urn:scte:scte35:2014:xml+bin">

```

```

    <Event duration="540000" id="144">
      <scte35:Signal>
        <scte35:Binary>SCTE35-binary</scte35:Binary>
      </scte35:Signal>
    </Event>
  </EventStream>
  ...
</Period>
<Period start="PT46921.928S" id="49">
  ...
</Period>
</MPD>

```

Example 2: MediaTailor personalized DASH manifest containing an ad ID decoration

```

<?xml version="1.0" encoding="utf-8"?>
<MPD xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="urn:mpeg:dash:schema:mpd:2011" xmlns:scte35="urn:scte:scte35:2013:xml"
  xsi:schemaLocation="urn:mpeg:dash:schema:mpd:2011 http://standards.iso.org/
  ittf/PubliclyAvailableStandards/MPEG-DASH_schema_files/DASH-MPD.xsd" id="201"
  type="dynamic" publishTime="2022-11-07T19:59:05+00:00" minimumUpdatePeriod="PT2S"
  availabilityStartTime="2022-11-07T06:57:11.250000+00:00" minBufferTime="PT10S"
  suggestedPresentationDelay="PT20.000S" timeShiftBufferDepth="PT58.999S"
  profiles="urn:mpeg:dash:profile:isoff-live:2011">
  <Period start="PT46827.601S" id="0" duration="PT88.321S">
    ...
  </Period>
  <Period start="PT46915.922S" id="45" duration="PT6.006S">
    <EventStream schemeIdUri="urn:sva:advertising-wg:ad-id-signaling" timescale="90000">
      <Event presentationTime="13500000" duration="1351350">
        <![CDATA[{"version": 1,"identifiers": [{"scheme":
"urn:smpte:ul:060E2B34.01040101.01200900.00000000","value": "adId","ad_position":
"adId", "ad_type":"overlay","creative_id": "creativeId","tracking_uri":
"trackingUri"}]]]></Event>
      </EventStream>
    ...
  </Period>
  <Period start="PT46921.928S" id="49">
    ...
  </Period>
</MPD>

```


Tracking overlay ads with client-side metadata

MediaTailor places the overlay ads in the `nonLinearAdsList` of the `avail`. The MediaTailor client-side tracking API has two root objects, called `avails` and `nonLinearAvails`. If the VAST response is a VMAP with `breakType` of `nonlinear`, the `avail` metadata is inside the `nonLinearAvails` root object. If the VAST response is a VMAP with a `breakType` of `linear`, or is a plain VAST response without VMAP, the `avail` metadata is inside the `avails` root object.

For more information about client-side tracking, see [Client-side ad tracking](#).

The following example shows a plain VAST response or VMAP response with a `breakType` value of `linear`.

```
{
  "avails": [
    {
      "adBreakTrackingEvents": [
        {
          "beaconUrls": [
            "https://adserver.com/beacon=breakstartimpression"
          ],
          "eventType": "breakStart"
        },
        {
          "beaconUrls": [
            "https://adserver.com/beacon=breakendimpression"
          ],
          "eventType": "breakEnd"
        }
      ],
      "adMarkerDuration": null,
      "ads": [],
      "availId": "828",
      "availProgramDateTime": null,
      "duration": "PT0S",
      "durationInSeconds": 0,
      "meta": null,
      "nonLinearAdsList": [
        {
          "extensions": null,
          "nonLinearAdList": [
            {
              "adId": "",

```

```

        "adParameters": null,
        "adSystem": "2.0",
        "adTitle": "2",
        "apiFramework": null,
        "clickThrough": null,
        "clickTracking": null,
        "clickTrackingId": null,
        "creativeAdId": "",
        "creativeId": "18",
        "creativeSequence": "",
        "duration": null,
        "durationInSeconds": 0,
        "expandedHeight": null,
        "expandedWidth": null,
        "height": "360",
        "htmlResource": null,
        "iFrameResource": null,
        "maintainAspectRatio": false,
        "minSuggestedDuration": null,
        "scalable": false,
        "staticResource": "https://client-side-ads.com/tags/static/ctv-generic/overlay001.json?iv_geo_country%3DUS%26",
        "staticResourceCreativeType": "text/js_ref",
        "width": "640"
    },
    "trackingEvents": [
        {
            "beaconUrls": [
                "https://adserver.com/beacon=impression"
            ],
            "duration": null,
            "durationInSeconds": 0,
            "eventId": null,
            "eventProgramDateTime": null,
            "eventType": "impression",
            "startTime": null,
            "startTimeInSeconds": 0
        }
    ]
},
"startTime": "PT1M46.08S",
"startTimeInSeconds": 106.08

```

```

    }
  ],
  "dashAvailabilityStartTime": null,
  "hlsAnchorMediaSequenceNumber": null,
  "nextToken": null,
  "nonLinearAvails": []
}

```

The following example shows a plain VMAP response with a breakType value of nonlinear.

```

{
  "avails": [],
  "dashAvailabilityStartTime": null,
  "hlsAnchorMediaSequenceNumber": null,
  "nextToken": null,
  "nonLinearAvails": [
    {
      "adBreakTrackingEvents": [
        {
          "beaconUrls": [
            "https://adserver.com/beacon=breakstartimpression"
          ],
          "eventType": "breakStart"
        },
        {
          "beaconUrls": [
            "https://adserver.com/beacon=breakendimpression"
          ],
          "eventType": "breakEnd"
        }
      ],
      "adMarkerDuration": null,
      "ads": [],
      "availId": "828",
      "availProgramDateTime": null,
      "duration": "PT0S",
      "durationInSeconds": 0,
      "meta": null,
      "nonLinearAdsList": [
        {
          "extensions": null,
          "nonLinearAdList": [

```

```

        "adId": "",
        "adParameters": null,
        "adSystem": "2.0",
        "adTitle": "2",
        "apiFramework": null,
        "clickThrough": null,
        "clickTracking": null,
        "clickTrackingId": null,
        "creativeAdId": "",
        "creativeId": "18",
        "creativeSequence": "",
        "duration": null,
        "durationInSeconds": 0,
        "expandedHeight": null,
        "expandedWidth": null,
        "height": "360",
        "htmlResource": null,
        "iFrameResource": null,
        "maintainAspectRatio": false,
        "minSuggestedDuration": null,
        "scalable": false,
        "staticResource": "https://client-side-ads.com/tags/static/ctv-generic/overlay001.json?iv_geo_country%3DUS%26",
        "staticResourceCreativeType": "text/js_ref",
        "width": "640"
    },
    ],
    "trackingEvents": [
        {
            "beaconUrls": [
                "https://adserver.com/beacon=impression"
            ],
            "duration": null,
            "durationInSeconds": 0,
            "eventId": null,
            "eventProgramDateTime": null,
            "eventType": "impression",
            "startTime": null,
            "startTimeInSeconds": 0
        }
    ]
},
],
"startTime": "PT1M46.08S",

```

```
    "startTimeInSeconds": 106.08
  }
]
}
```

Logging and metrics for overlay ads in MediaTailor

This section explains logging and metrics for overlay ads in MediaTailor. For more information about setting up logging, see [Monitoring and tagging AWS Elemental MediaTailor resources](#).

Topics

- [CloudWatch logs](#)
- [CloudWatch metrics](#)

CloudWatch logs

CloudWatch collects the following log information about overlay ads:

- VAST_RESPONSE - Shows information about the non-linear ads list.
- FILLED_PROVIDER_OVERLAY - Shows information about the non-linear ads.

Note

The RAW_ADS_RESPONSE is an optional event that shows the original response from the ADS. Using this event is especially helpful in a staging and testing environment. To enable this event on a configuration or account, submit a ticket to AWS Support.

CloudWatch metrics

MediaTailor collects overlay ad metrics separately from other ADS metrics. MediaTailor collects these metrics after successfully fetching the ads from the ADS. You don't have to poll the GetTracking API to collect the metrics.

The following table describes CloudWatch metrics for overlay ads:

Metric	Description
<code>AdDecisionServer.OverlayAds</code>	The count of overlay ads included in the ADS responses within the CloudWatch time period that you specified.
<code>AdDecisionServer.OverlayErrors</code>	The number of non-HTTP 200 status code responses, empty responses, and timed-out responses that MediaTailor received from the ADS within the CloudWatch time period that you specified.
<code>AdDecisionServer.OverlayFilled</code>	The number of avails that were successfully filled with at least one overlay ad: • 1 - There's at least one valid ad. • 0 - Either MediaTailor didn't get any overlay ads, or there was some other failure. <code>SampleCount</code> tracks the number of avails filled. <code>Sum</code> tracks the number of successfully filled overlay avails.
<code>AdDecisionServer.OverlayMinSuggestedDuration</code>	The sum of <code>minSuggestedDuration</code> durations, in milliseconds, of all ads that MediaTailor received from the ADS within the CloudWatch time period that you specified. If <code>minSuggestedDuration</code> isn't specified, the duration shown is the planned duration.
<code>AdDecisionServer.OverlayLatency</code>	The response time, in milliseconds, for requests that MediaTailor makes to the ADS.

Metric	Description
<code>AdDecisionServer.OverlayTimeouts</code>	The number of timed-out requests to the ADS in the CloudWatch time period that you specified.
<code>AdsBilled</code>	For more information about ads billed, see Billing for overlay ads in MediaTailor .
<code>Avail.*</code>	Because MediaTailor doesn't do any planning for overlay ads, CloudWatch doesn't show any <code>Avail.X</code> metrics.
<code>SkippedReason.*</code>	Because MediaTailor doesn't do any planning for overlay ads, CloudWatch doesn't show any <code>SkippedReason.X</code> metrics.

Billing for overlay ads in MediaTailor

MediaTailor bills customers based on the number of non-linear ads in the ADS response. This number includes non-linear ads that extend past the break duration. After MediaTailor fills the avail, it bills for the ads it filled.

For prefetch workflows, MediaTailor does not bill for ads when retrieving the prefetch, but rather, when it sees a compatible ad avail in the consumption window for that session.

For additional billing information, see <https://aws.amazon.com/mediatailor/pricing/>.

Ad ID decoration

AWS Elemental MediaTailor performs server side ad stitching when transitioning from content to ad breaks. MediaTailor can condition the manifest with metadata associated with the ads that have been stitched. Doing so can provide the following benefits:

⚠ Important

Ad-ID decoration is not compatible with server-guided ad insertion (SGAI) methods because the fields that populate X-AD-CREATIVE-SIGNALING headers are only known when the asset list is fetched during ad playback, not when the cacheable manifest is initially written.

- *Video start time (VST)* improves
- MediaTailor can support a hybrid model of server side ad insertion and server guided ad insertion
- Server side sessions can build playback timelines with ad position markers
- For client side sessions that already build playback timelines with the MediaTailor API, session VST improves, as the session does not rely on calling the tracking API to build the timeline
- It's possible to leverage MediaTailor for server side ad insertion as well as client side rendered ads displayed in-scene. This way, a player's software development kit (SDK) doesn't need to have a separate integration to call ad serving entities directly for client-side ads. MediaTailor can vend the ads through the manifest and the client-side tracking API.

There are standards for associating each creative ad asset with a unique identifier. This association allows advertisers, agencies, vendors, and publishers to relate a creative ad asset across their independent workflows. As metrics and monitoring of streams continue to improve and more distributors utilize server-based insertion architectures, the need arises to accurately communicate the identifiers assigned to individual creative assets within an interleaved/stitched presentation, such as within the personalized manifest.

Topics

- [Enabling ad ID signaling for sessions](#)
- [Manifests and ad metadata insertion](#)
- [Ad Decision Server \(ADS\) interactions](#)
- [Client-side tracking API](#)

Enabling ad ID signaling for sessions

The ad ID signaling feature must be enabled during session initialization. The process to enable the feature differs from creating sessions using the HLS/DASH playback prefix (implicit session initialization), versus the session initialization prefix (explicit session initialization).

To enable ad ID for the session using HLS/DASH playback prefixes

- From the player, initialize a new MediaTailor playback session using a request in one of the following formats, according to your protocol:

- Example: HLS format

```
GET <mediatailorURL>/v1/master/<hashed-account-id>/<origin-id>/<asset-id>?
aws.adSignalingEnabled=true
```

- Example: DASH format

```
GET <mediatailorURL>/v1/dash/<hashed-account-id>/<origin-id>/<asset-id>?
aws.adSignalingEnabled=true
```

To enable ad ID for the session using the session initialization prefix

- On the player, construct a JSON message body for the session initialization request to MediaTailor:
 - Inside an `adsParams` object, provide any parameters that MediaTailor should pass to the ADS. These parameters correspond to `[player_params.param]` settings in the ADS template URL of the MediaTailor configuration.
 - To enable ad ID signaling, add an `adSignaling` object as a top level object, and inside, add a parameter called `enabled` and value of `true`. The default `adSignaling` value is disabled.

- Example: HLS format

```
POST master.m3u8
{
  "adsParams": {
    "deviceType": "ipad"
```

```
    },  
    "adSignaling": {  
        "enabled": "true"  
    },  
    "reportingMode": "client"  
}
```

- Example: DASH format

```
POST manifest.mpd  
{  
    "adsParams": {  
        "deviceType": "ipad"  
    },  
    "adSignaling": {  
        "enabled": "true"  
    },  
    "reportingMode": "client"  
}
```

Manifests and ad metadata insertion

During the ad stitching process, MediaTailor adds to the manifest the unique ID associated with each creative being stitched. MediaTailor obtains the unique ID of the creative from the `id` attribute value of that creative in the VAST response. If the creative lacks an ID attribute value, MediaTailor will publish an empty value (`id=""`).

MediaTailor uses an in-manifest metadata signal to decouple dependencies between the client tracking API for ad creative metadata and timing/positioning within the overall timeline. This decoupling reduces playback latency (especially in VOD scenarios), where the player's user interface (UI) renders ad break positions in the timeline prior to initializing playback.

The added metadata takes the following forms:

- For HLS manifests, the added metadata takes the form of DATERANGE tags for each ad in the avail period.
- For DASH manifests, the added metadata takes the form of an Event element within each ad period.

The following JSON message body shows an example VAST response:

```
{
  "version": 1,
  "identifiers": [
    {
      "scheme": "urn:smp:ul:060E2B34.01040101.01200900.00000000",
      "value": "creativeId",
      "ad_position": "adId",
      "ad_type": "adType",
      "tracking_uri": "trackingUri",
      "custom_vast_data": "customVastData"
    }
  ]
}
```

In the preceding example:

- *creativeId* is the Id attribute value of the Creative element for the ad
- *adId* is either the HLS sequence number associated with the beginning of the ad, or the DASH period ID of the ad
- *adType* is either avail or overlay, based on the VAST response
- *trackingUri* is the relative tracking endpoint for the MediaTailor session, in the format `../../../../tracking/hashed-account-id/origin-id/session-id`
- *customVastData* is a value that MediaTailor extracts from the creative_signaling VAST extension. MediaTailor uses the contents of the CDATA node, if present. See the [Ad Decision Server \(ADS\) interactions](#) section for more details and a sample VAST response.

Personalizing HLS manifests with ad metadata

For a live HLS stream, MediaTailor only adds metadata when the stream contains PROGRAM-DATE-TIME tags, at least once per manifest duration. For a video on demand (VOD) stream, MediaTailor adds PROGRAM-DATE-TIME to at least one segment in the personalized manifest, where the start time for each VOD asset is epoch zero (1970-01-01T00:00:00Z). If the origin manifest has existing PROGRAM-DATE-TIME content, then MediaTailor preserves that content.

MediaTailor personalizes the manifest with creatives returned by the Ad Decision Server (ADS). For each ad, MediaTailor also includes a DATERANGE tag that spans the duration of the ad. The

DATERANGE tag format is similar to that described in the section [Ad creative signaling in DASH and HLS](#) in the 2023 version of the *SVA technical publication*.

The DATERANGE that MediaTailor generates has unique ID values. To ensure uniqueness (given the guidelines specified in [Mapping SCTE-35 into EXT-X-DATERANGE](#)), MediaTailor couples the MEDIA-SEQUENCE number of the *first* ad segment of the avail with the sequence number of the ad within the avail.

For underfilled ad breaks on configurations that have slate enabled, MediaTailor appends the slate segments to the end of the avail, separated by a DISCONTINUITY tag, but without any DATERANGE metadata.

For each ad stitched into the personalized manifest, MediaTailor adds the creative metadata, represented as base64-encoded data in a custom DATERANGE tag.

Example Linear HLS origin (#EXT-X-CUE-OUT):

```
#EXTM3U
#EXT-X-VERSION:3
#EXT-X-TARGETDURATION:7
#EXT-X-MEDIA-SEQUENCE:398
#EXT-X-PROGRAM-DATE-TIME:2023-02-10T19:20:01.397Z
#EXTINF:6.006,
index_1_398.ts?m=1676054627
#EXTINF:5.873,
index_1_399.ts?m=1676054627
#EXT-OATCLS-SCTE35:/DA1AAAAAyIYAP/wFAUAAAACf+//jP197P4AUmNiAAEBAQAase4/gA==
#EXT-X-CUE-OUT:59.993
#EXTINF:6.139,
index_1_400.ts?m=1676054627
#EXT-X-CUE-OUT-CONT:ElapsedTime=6.139,Duration=59.993,SCTE35=/DA1AAAAAyIYAP/wFAUAAAACf+//jP197P4AUmNiAAEBAQAase4/gA==
#EXTINF:6.006,
index_1_401.ts?m=1676054627
#EXT-X-CUE-OUT-CONT:ElapsedTime=12.145,Duration=59.993,SCTE35=/DA1AAAAAyIYAP/wFAUAAAACf+//jP197P4AUmNiAAEBAQAase4/gA==
#EXTINF:6.006,
index_1_402.ts?m=1676054627
#EXT-X-CUE-OUT-CONT:ElapsedTime=18.151,Duration=59.993,SCTE35=/DA1AAAAAyIYAP/wFAUAAAACf+//jP197P4AUmNiAAEBAQAase4/gA==
#EXTINF:6.006,
index_1_403.ts?m=1676054627
```

```
#EXT-X-CUE-OUT-CONT:ElapsedTime=24.157,Duration=59.993,SCTE35=/DA1AAAAAyiYAP/wFAUAAAACf
+//jP197P4AUmNiAAEBAQAase4/gA==
#EXTINF:6.006,
index_1_404.ts?m=1676054627
#EXT-X-CUE-OUT-CONT:ElapsedTime=30.163,Duration=59.993,SCTE35=/DA1AAAAAyiYAP/wFAUAAAACf
+//jP197P4AUmNiAAEBAQAase4/gA==
#EXTINF:6.006,
index_1_405.ts?m=1676054627
#EXT-X-CUE-OUT-CONT:ElapsedTime=36.169,Duration=59.993,SCTE35=/DA1AAAAAyiYAP/wFAUAAAACf
+//jP197P4AUmNiAAEBAQAase4/gA==
#EXTINF:6.006,
index_1_406.ts?m=1676054627
#EXT-X-CUE-OUT-CONT:ElapsedTime=42.175,Duration=59.993,SCTE35=/DA1AAAAAyiYAP/wFAUAAAACf
+//jP197P4AUmNiAAEBAQAase4/gA==
#EXTINF:6.006,
index_1_407.ts?m=1676054627
#EXT-X-CUE-OUT-CONT:ElapsedTime=48.181,Duration=59.993,SCTE35=/DA1AAAAAyiYAP/wFAUAAAACf
+//jP197P4AUmNiAAEBAQAase4/gA==
#EXTINF:6.006,
index_1_408.ts?m=1676054627
#EXT-X-CUE-OUT-CONT:ElapsedTime=54.187,Duration=59.993,SCTE35=/DA1AAAAAyiYAP/wFAUAAAACf
+//jP197P4AUmNiAAEBAQAase4/gA==
#EXTINF:5.806,
index_1_409.ts?m=1676054627
#EXT-X-CUE-IN
#EXTINF:6.206,
index_1_410.ts?m=1676054627
#EXTINF:6.006,
index_1_411.ts?m=1676054627
#EXTINF:6.006,
index_1_412.ts?m=1676054627
```

Example Linear HLS origin (#EXT-X-DATERANGE):

```
#EXTM3U
#EXT-X-VERSION:3
#EXT-X-TARGETDURATION:7
#EXT-X-MEDIA-SEQUENCE:25
#EXT-X-PROGRAM-DATE-TIME:2023-02-10T19:19:53.389Z
#EXTINF:6.006,
index_1_25.ts?m=1676056675
#EXTINF:6.006,
index_1_26.ts?m=1676056675
```

```
#EXTINF:6.006,
index_1_27.ts?m=1676056675
#EXTINF:1.869,
index_1_28.ts?m=1676056675
#EXT-X-DATERANGE:ID="2",START-DATE="2023-02-10T19:20:13.276Z",PLANNED-
DURATION=59.993,SCTE35-
OUT=0xFC302500000003289800FFF01405000000027FEFFF8CF97DECFE00526362000101010000B1EE3F80
#EXTINF:6.139,
index_1_29.ts?m=1676056675
#EXTINF:6.006,
index_1_30.ts?m=1676056675
#EXTINF:6.006,
index_1_31.ts?m=1676056675
#EXTINF:6.006,
index_1_32.ts?m=1676056675
#EXTINF:6.006,
index_1_33.ts?m=1676056675
#EXTINF:6.006,
index_1_34.ts?m=1676056675
#EXTINF:6.006,
index_1_35.ts?m=1676056675
#EXTINF:6.006,
index_1_36.ts?m=1676056675
#EXTINF:6.006,
index_1_37.ts?m=1676056675
#EXTINF:5.806,
index_1_38.ts?m=1676056675
#EXT-X-DATERANGE:ID="2",START-DATE="2023-02-10T19:20:13.276Z",END-
DATE="2023-02-10T19:21:13.269Z",DURATION=59.993
#EXTINF:6.206,
index_1_39.ts?m=1676056675
#EXTINF:6.006,
index_1_40.ts?m=1676056675
```

Example Linear HLS personalized manifest (with creative ad signaling):

The DATERANGE that MediaTailor generates has unique ID values. To ensure uniqueness (given the guidelines specified in [Mapping SCTE-35 into EXT-X-DATERANGE](#)), MediaTailor couples the MEDIA-SEQUENCE number of the *first* ad segment of the avail with the sequence number of the ad within the avail.

In the following example, MediaTailor concatenates MEDIA-SEQUENCE 421 with the ad position number.

```

#EXTM3U
#EXT-X-VERSION:6
#EXT-X-TARGETDURATION:7
#EXT-X-MEDIA-SEQUENCE:418
#EXT-X-DISCONTINUITY-SEQUENCE:5
#EXT-X-PROGRAM-DATE-TIME:2023-02-10T19:19:55.391Z
#EXTINF:6.006,
https://d3fch9e2fcarly.cloudfront.net/out/v1/1cc7058242a74fdd8aea14e22a9b4131/
index_1_397.ts?m=1676054627
#EXTINF:6.006,
https://d3fch9e2fcarly.cloudfront.net/out/v1/1cc7058242a74fdd8aea14e22a9b4131/
index_1_398.ts?m=1676054627
#EXTINF:5.873,
https://d3fch9e2fcarly.cloudfront.net/out/v1/1cc7058242a74fdd8aea14e22a9b4131/
index_1_399.ts?m=1676054627
#EXT-X-DISCONTINUITY
#EXT-X-PROGRAM-DATE-TIME:2023-02-10T19:19:55.391Z
#EXT-X-DATERANGE:ID="421-1",CLASS="urn:sva:advertising-wg:ad-id-signaling",START-
DATE=2019-01-01T00:02:30.000Z,DURATION=15.015,X-AD-CREATIVE-SIGNALING="base64JSON"
#EXTINF:2.002,
../../../../segment/94063eadf7d8c56e9e2edd84fdf897826a70d0df/emt/9e178fa9-
dce5-4248-83d2-5b5d98b019bf/0/1676056813
#EXTINF:2.002,
../../../../segment/94063eadf7d8c56e9e2edd84fdf897826a70d0df/emt/9e178fa9-
dce5-4248-83d2-5b5d98b019bf/0/1676056814
#EXTINF:2.002,
../../../../segment/94063eadf7d8c56e9e2edd84fdf897826a70d0df/emt/9e178fa9-
dce5-4248-83d2-5b5d98b019bf/0/1676056815
#EXTINF:2.002,
../../../../segment/94063eadf7d8c56e9e2edd84fdf897826a70d0df/emt/9e178fa9-
dce5-4248-83d2-5b5d98b019bf/0/1676056816
#EXTINF:2.002,
../../../../segment/94063eadf7d8c56e9e2edd84fdf897826a70d0df/emt/9e178fa9-
dce5-4248-83d2-5b5d98b019bf/0/1676056817
#EXTINF:2.002,
../../../../segment/94063eadf7d8c56e9e2edd84fdf897826a70d0df/emt/9e178fa9-
dce5-4248-83d2-5b5d98b019bf/0/1676056818
#EXTINF:2.002,
../../../../segment/94063eadf7d8c56e9e2edd84fdf897826a70d0df/emt/9e178fa9-
dce5-4248-83d2-5b5d98b019bf/0/1676056819
#EXTINF:1.001,
../../../../segment/94063eadf7d8c56e9e2edd84fdf897826a70d0df/emt/9e178fa9-
dce5-4248-83d2-5b5d98b019bf/0/1676056820

```

```

#EXT-X-DISCONTINUITY
#EXT-X-PROGRAM-DATE-TIME:2023-02-10T19:19:55.391Z
#EXT-X-DATERANGE:ID="421-1",START-DATE="2023-02-10T19:36:13.435Z",END-
DATE="2023-02-10T19:36:43.432Z",DURATION=15.015
#EXT-X-DATERANGE:ID="421-2",CLASS="urn:sva:advertising-wg:ad-id-signaling",START-
DATE=2019-01-01T00:02:30.000Z,DURATION=15.015,X-AD-CREATIVE-SIGNALING="base64JSON"
#EXTINF:2.002,
../././././segment/94063eadf7d8c56e9e2edd84fdf897826a70d0df/emt/9e178fa9-
dce5-4248-83d2-5b5d98b019bf/0/1676056821
#EXTINF:2.002,
../././././segment/94063eadf7d8c56e9e2edd84fdf897826a70d0df/emt/9e178fa9-
dce5-4248-83d2-5b5d98b019bf/0/1676056822
#EXTINF:2.002,
../././././segment/94063eadf7d8c56e9e2edd84fdf897826a70d0df/emt/9e178fa9-
dce5-4248-83d2-5b5d98b019bf/0/1676056823
#EXTINF:2.002,
../././././segment/94063eadf7d8c56e9e2edd84fdf897826a70d0df/emt/9e178fa9-
dce5-4248-83d2-5b5d98b019bf/0/1676056824
#EXTINF:2.002,
../././././segment/94063eadf7d8c56e9e2edd84fdf897826a70d0df/emt/9e178fa9-
dce5-4248-83d2-5b5d98b019bf/0/1676056825
#EXTINF:2.002,
../././././segment/94063eadf7d8c56e9e2edd84fdf897826a70d0df/emt/9e178fa9-
dce5-4248-83d2-5b5d98b019bf/0/1676056826
#EXTINF:2.002,
../././././segment/94063eadf7d8c56e9e2edd84fdf897826a70d0df/emt/9e178fa9-
dce5-4248-83d2-5b5d98b019bf/0/1676056827
#EXTINF:1.001,
../././././segment/94063eadf7d8c56e9e2edd84fdf897826a70d0df/emt/9e178fa9-
dce5-4248-83d2-5b5d98b019bf/0/1676056828
#EXT-X-DISCONTINUITY
#EXT-X-PROGRAM-DATE-TIME:2023-02-10T19:19:55.391Z
#EXT-X-DATERANGE:ID="421-2",START-DATE="2023-02-10T19:36:13.435Z",END-
DATE="2023-02-10T19:36:43.432Z",DURATION=15.015
#EXT-X-DATERANGE:ID="421-3",CLASS="urn:sva:advertising-wg:ad-id-signaling",START-
DATE=2019-01-01T00:02:30.000Z,DURATION=15.015,X-AD-CREATIVE-SIGNALING="base64JSON"
#EXTINF:2.002,
../././././segment/94063eadf7d8c56e9e2edd84fdf897826a70d0df/emt/9e178fa9-
dce5-4248-83d2-5b5d98b019bf/0/1676056829
#EXTINF:2.002,
../././././segment/94063eadf7d8c56e9e2edd84fdf897826a70d0df/emt/9e178fa9-
dce5-4248-83d2-5b5d98b019bf/0/1676056830
#EXTINF:2.002,

```



```

../././././segment/94063eadf7d8c56e9e2edd84fdf897826a70d0df/emt/9e178fa9-
dce5-4248-83d2-5b5d98b019bf/0/1676056831
#EXTINF:2.002,
../././././segment/94063eadf7d8c56e9e2edd84fdf897826a70d0df/emt/9e178fa9-
dce5-4248-83d2-5b5d98b019bf/0/1676056832
#EXTINF:2.002,
../././././segment/94063eadf7d8c56e9e2edd84fdf897826a70d0df/emt/9e178fa9-
dce5-4248-83d2-5b5d98b019bf/0/1676056833
#EXTINF:2.002,
../././././segment/94063eadf7d8c56e9e2edd84fdf897826a70d0df/emt/9e178fa9-
dce5-4248-83d2-5b5d98b019bf/0/1676056834
#EXTINF:2.002,
../././././segment/94063eadf7d8c56e9e2edd84fdf897826a70d0df/emt/9e178fa9-
dce5-4248-83d2-5b5d98b019bf/0/1676056835
#EXTINF:1.001,
../././././segment/94063eadf7d8c56e9e2edd84fdf897826a70d0df/emt/9e178fa9-
dce5-4248-83d2-5b5d98b019bf/0/1676056836
#EXT-X-DISCONTINUITY
#EXT-X-PROGRAM-DATE-TIME:2023-02-10T19:19:55.391Z
#EXT-X-DATERANGE:ID="421-3",START-DATE="2023-02-10T19:36:13.435Z",END-
DATE="2023-02-10T19:36:43.432Z",DURATION=29.997
#EXT-X-DATERANGE:ID="421-4",CLASS="urn:sva:advertising-wg:ad-id-signaling",START-
DATE=2019-01-01T00:02:30.000Z,DURATION=15.015,X-AD-CREATIVE-SIGNALING="base64JSON"
#EXTINF:2.002,
../././././segment/94063eadf7d8c56e9e2edd84fdf897826a70d0df/emt/9e178fa9-
dce5-4248-83d2-5b5d98b019bf/0/1676056837
#EXTINF:2.002,
../././././segment/94063eadf7d8c56e9e2edd84fdf897826a70d0df/emt/9e178fa9-
dce5-4248-83d2-5b5d98b019bf/0/1676056838
#EXTINF:2.002,
../././././segment/94063eadf7d8c56e9e2edd84fdf897826a70d0df/emt/9e178fa9-
dce5-4248-83d2-5b5d98b019bf/0/1676056839
#EXTINF:2.002,
../././././segment/94063eadf7d8c56e9e2edd84fdf897826a70d0df/emt/9e178fa9-
dce5-4248-83d2-5b5d98b019bf/0/1676056840
#EXTINF:2.002,
../././././segment/94063eadf7d8c56e9e2edd84fdf897826a70d0df/emt/9e178fa9-
dce5-4248-83d2-5b5d98b019bf/0/1676056841
#EXTINF:2.002,
../././././segment/94063eadf7d8c56e9e2edd84fdf897826a70d0df/emt/9e178fa9-
dce5-4248-83d2-5b5d98b019bf/0/1676056842
#EXTINF:2.002,
../././././segment/94063eadf7d8c56e9e2edd84fdf897826a70d0df/emt/9e178fa9-
dce5-4248-83d2-5b5d98b019bf/0/1676056843

```

```
#EXTINF:1.001,  
../.../segment/94063eadf7d8c56e9e2edd84fdf897826a70d0df/emt/9e178fa9-  
dce5-4248-83d2-5b5d98b019bf/0/1676056844  
#EXT-X-DISCONTINUITY  
#EXT-X-DATERANGE:ID="421-4",START-DATE="2023-02-10T19:36:13.435Z",END-  
DATE="2023-02-10T19:36:43.432Z",DURATION=15.015  
#EXTINF:6.206,  
https://d3fch9e2fcarly.cloudfront.net/out/v1/1cc7058242a74fdd8aea14e22a9b4131/  
index_1_410.ts?m=1676054627  
#EXTINF:6.006,  
https://d3fch9e2fcarly.cloudfront.net/out/v1/1cc7058242a74fdd8aea14e22a9b4131/  
index_1_411.ts?m=1676054627
```

Example VOD HLS origin (with SCTE signals):

```
#EXTM3U  
#EXT-X-VERSION:3  
#EXT-X-TARGETDURATION:7  
#EXT-X-MEDIA-SEQUENCE:1  
#EXT-X-PLAYLIST-TYPE:VOD  
#EXTINF:6,  
index_720p1500k_00001.ts  
#EXTINF:6,  
index_720p1500k_00002.ts  
#EXTINF:6,  
index_720p1500k_00003.ts  
#EXTINF:6,  
index_720p1500k_00004.ts  
#EXTINF:6,  
index_720p1500k_00005.ts  
#EXT-X-CUE-OUT:0  
#EXT-X-CUE-IN  
#EXTINF:6,  
index_720p1500k_00006.ts  
#EXTINF:6,  
index_720p1500k_00007.ts  
#EXTINF:6,  
index_720p1500k_00008.ts  
#EXTINF:6,  
index_720p1500k_00009.ts  
#EXTINF:6,  
index_720p1500k_00010.ts  
#EXTINF:6,
```

```
index_720p1500k_00011.ts
#EXTINF:6,
index_720p1500k_00012.ts
```

Example VOD HLS origin:

```
#EXTM3U
#EXT-X-VERSION:3
#EXT-X-TARGETDURATION:7
#EXT-X-MEDIA-SEQUENCE:1
#EXT-X-PLAYLIST-TYPE:VOD
#EXTINF:6,
index_720p1500k_00001.ts
#EXTINF:6,
index_720p1500k_00002.ts
#EXTINF:6,
index_720p1500k_00003.ts
#EXTINF:6,
index_720p1500k_00004.ts
#EXTINF:4,
index_720p1500k_00005.ts
#EXTINF:2,
index_720p1500k_00006.ts
#EXTINF:6,
index_720p1500k_00007.ts
#EXTINF:6,
index_720p1500k_00008.ts
#EXTINF:6,
index_720p1500k_00009.ts
#EXTINF:6,
index_720p1500k_00010.ts
#EXTINF:6,
index_720p1500k_00011.ts
#EXTINF:6,
index_720p1500k_00012.ts
```

Example VOD HLS personalized manifest:

MediaTailor adds PROGRAM-DATE-TIME to VOD manifests in order to use them as anchors for the HLS DATERANGE elements that indicate ad positions.

The DATERANGE that MediaTailor generates has unique ID values. To ensure uniqueness (given the guidelines specified in [Mapping SCTE-35 into EXT-X-DATERANGE](#)), MediaTailor couples the MEDIA-

SEQUENCE number of the *first* ad segment of the avail with the sequence number of the ad within the avail.

In the following example, MediaTailor concatenates MEDIA-SEQUENCE 421 with the ad position number.

```
#EXTM3U
#EXT-X-VERSION:3
#EXT-X-PLAYLIST-TYPE:VOD
#EXT-X-TARGETDURATION:7
#EXT-X-MEDIA-SEQUENCE:1
#EXT-X-DISCONTINUITY-SEQUENCE:0
#EXT-X-PROGRAM-DATE-TIME:1970-01-01T00:00:00Z
#EXTINF:6.0,
https://d3fch9e2fcarly.cloudfront.net/cunsco-media/SK0-22/asset-1/hls/
index_720p1500k_00001.ts
#EXTINF:6.0,
https://d3fch9e2fcarly.cloudfront.net/cunsco-media/SK0-22/asset-1/hls/
index_720p1500k_00002.ts
#EXTINF:6.0,
https://d3fch9e2fcarly.cloudfront.net/cunsco-media/SK0-22/asset-1/hls/
index_720p1500k_00003.ts
#EXTINF:6.0,
https://d3fch9e2fcarly.cloudfront.net/cunsco-media/SK0-22/asset-1/hls/
index_720p1500k_00004.ts
#EXTINF:4.0,
https://d3fch9e2fcarly.cloudfront.net/cunsco-media/SK0-22/asset-1/hls/
index_720p1500k_00005.ts
#EXT-X-DISCONTINUITY
#EXTINF:2.002,
../../../../segment/94063eadf7d8c56e9e2edd84fdf897826a70d0df/vod-
variations/9810d863-8736-45fa-866e-be6d2c2bfa20/0/28
#EXTINF:2.002,
../../../../segment/94063eadf7d8c56e9e2edd84fdf897826a70d0df/vod-
variations/9810d863-8736-45fa-866e-be6d2c2bfa20/0/29
#EXTINF:2.002,
../../../../segment/94063eadf7d8c56e9e2edd84fdf897826a70d0df/vod-
variations/9810d863-8736-45fa-866e-be6d2c2bfa20/0/30
#EXTINF:2.002,
../../../../segment/94063eadf7d8c56e9e2edd84fdf897826a70d0df/vod-
variations/9810d863-8736-45fa-866e-be6d2c2bfa20/0/31
#EXTINF:2.002,
```

```

../.../segment/94063eadf7d8c56e9e2edd84fdf897826a70d0df/vod-
variations/9810d863-8736-45fa-866e-be6d2c2bfa20/0/32
#EXTINF:2.002,
../.../segment/94063eadf7d8c56e9e2edd84fdf897826a70d0df/vod-
variations/9810d863-8736-45fa-866e-be6d2c2bfa20/0/33
#EXTINF:2.002,
../.../segment/94063eadf7d8c56e9e2edd84fdf897826a70d0df/vod-
variations/9810d863-8736-45fa-866e-be6d2c2bfa20/0/34
#EXTINF:1.001,
../.../segment/94063eadf7d8c56e9e2edd84fdf897826a70d0df/vod-
variations/9810d863-8736-45fa-866e-be6d2c2bfa20/0/35
#EXT-X-DISCONTINUITY
#EXTINF:2.002,
../.../segment/94063eadf7d8c56e9e2edd84fdf897826a70d0df/vod-
variations/9810d863-8736-45fa-866e-be6d2c2bfa20/0/36
#EXTINF:2.002,
../.../segment/94063eadf7d8c56e9e2edd84fdf897826a70d0df/vod-
variations/9810d863-8736-45fa-866e-be6d2c2bfa20/0/37
#EXTINF:2.002,
../.../segment/94063eadf7d8c56e9e2edd84fdf897826a70d0df/vod-
variations/9810d863-8736-45fa-866e-be6d2c2bfa20/0/38
#EXTINF:2.002,
../.../segment/94063eadf7d8c56e9e2edd84fdf897826a70d0df/vod-
variations/9810d863-8736-45fa-866e-be6d2c2bfa20/0/39
#EXTINF:2.002,
../.../segment/94063eadf7d8c56e9e2edd84fdf897826a70d0df/vod-
variations/9810d863-8736-45fa-866e-be6d2c2bfa20/0/40
#EXTINF:2.002,
../.../segment/94063eadf7d8c56e9e2edd84fdf897826a70d0df/vod-
variations/9810d863-8736-45fa-866e-be6d2c2bfa20/0/41
#EXTINF:2.002,
../.../segment/94063eadf7d8c56e9e2edd84fdf897826a70d0df/vod-
variations/9810d863-8736-45fa-866e-be6d2c2bfa20/0/42
#EXTINF:1.001,
../.../segment/94063eadf7d8c56e9e2edd84fdf897826a70d0df/vod-
variations/9810d863-8736-45fa-866e-be6d2c2bfa20/0/43
#EXT-X-DISCONTINUITY
#EXTINF:2.0,
https://d3fch9e2fcarly.cloudfront.net/cunsco-media/SK0-22/asset-1/hls/
index_720p1500k_00006.ts
#EXTINF:6.0,
https://d3fch9e2fcarly.cloudfront.net/cunsco-media/SK0-22/asset-1/hls/
index_720p1500k_00007.ts
#EXTINF:6.0,

```

```

https://d3fch9e2fcarly.cloudfront.net/cunsco-media/SK0-22/asset-1/hls/
index_720p1500k_00008.ts
#EXTINF:6.0,
https://d3fch9e2fcarly.cloudfront.net/cunsco-media/SK0-22/asset-1/hls/
index_720p1500k_00009.ts
#EXTINF:6.0,
https://d3fch9e2fcarly.cloudfront.net/cunsco-media/SK0-22/asset-1/hls/
index_720p1500k_00010.ts
#EXTINF:6.0,
https://d3fch9e2fcarly.cloudfront.net/cunsco-media/SK0-22/asset-1/hls/
index_720p1500k_00011.ts
#EXTINF:6.0,
https://d3fch9e2fcarly.cloudfront.net/cunsco-media/SK0-22/asset-1/hls/
index_720p1500k_00012.ts
#EXT-X-ENDLIST
#EXT-X-DATERANGE:ID="5-1",START-DATE="1970-01-01T00:00:28.000Z",END-
DATE="1970-01-01T00:00:43.015Z",DURATION=15.015
#EXT-X-DATERANGE:ID="5-2",START-DATE="1970-01-01T00:00:43.015Z",END-
DATE="1970-01-01T00:00:58.030Z",DURATION=15.01

```

Personalizing DASH manifests with ad metadata

MediaTailor personalizes the manifest with creatives returned by the Ad Decision Server (ADS). For each ad, MediaTailor also includes an `EventStream` element that spans the duration of the ad.

The `Event` element format is similar to that described in the section [Ad creative signaling in DASH and HLS](#) in the 2023 version of the *SVA technical publication*.

For underfilled ad breaks on configurations that have slate enabled, MediaTailor appends the slate period to the end of the avail period, but without any `EventStream` metadata

For each ad stitched into the personalized manifest, MediaTailor adds the creative metadata, represented as a `CDATA` element within an `Event` element.

Example Linear DASH origin (Inline SCTE attributes):

```

<MPD xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="urn:mpeg:dash:schema:mpd:2011" xmlns:scte35="urn:scte:scte35:2013:xml"
  xsi:schemaLocation="urn:mpeg:dash:schema:mpd:2011 http://standards.iso.org/
  ittf/PubliclyAvailableStandards/MPEG-DASH_schema_files/DASH-MPD.xsd" id="201"
  type="dynamic" publishTime="2023-02-10T21:08:40+00:00" minimumUpdatePeriod="PT6S"
  availabilityStartTime="2023-02-09T22:47:05.865000+00:00" minBufferTime="PT10S"
  suggestedPresentationDelay="PT20.000S" timeShiftBufferDepth="PT88.999S"
  profiles="urn:mpeg:dash:profile:isoff-live:2011">

```

```

<Period start="PT80141.456S" id="104" duration="PT304.103S">
  <AdaptationSet id="1485523442" mimeType="video/mp4" segmentAlignment="true"
startWithSAP="1" subsegmentAlignment="true" subsegmentStartsWithSAP="1"
bitstreamSwitching="true">
    <SegmentTemplate timescale="60000" media="index_video_$RepresentationID$_0_
$Number$.mp4?m=1676062374" initialization="index_video_$RepresentationID$_0_init.mp4?
m=1676062374" startNumber="151" presentationTimeOffset="4808487386">
        <SegmentTimeline>
            <S t="4824975858" d="360360" r="3"/>
            <S t="4826417298" d="316316"/>
        </SegmentTimeline>
    </SegmentTemplate>
    <Representation id="1" width="960" height="540" frameRate="30000/1001"
bandwidth="1800000" codecs="avc1.4D401F"/>
    <Representation id="3" width="640" height="360" frameRate="30000/1001"
bandwidth="1200000" codecs="avc1.4D401E"/>
    <Representation id="5" width="480" height="270" frameRate="30000/1001"
bandwidth="800000" codecs="avc1.4D4015"/>
  </AdaptationSet>
  <AdaptationSet id="1377232898" mimeType="audio/mp4" segmentAlignment="0"
lang="eng">
    <Label>eng</Label>
    <SegmentTemplate timescale="48000" media="index_audio_$RepresentationID$_0_
$Number$.mp4?m=1676062374" initialization="index_audio_$RepresentationID$_0_init.mp4?
m=1676062374" startNumber="151" presentationTimeOffset="3846790126">
        <SegmentTimeline>
            <S t="3859981294" d="287744"/>
            <S t="3860269038" d="288768"/>
            <S t="3860557806" d="287744"/>
            <S t="3860845550" d="288768"/>
            <S t="3861134318" d="252928"/>
        </SegmentTimeline>
    </SegmentTemplate>
    <Representation id="2" bandwidth="193007" audioSamplingRate="48000"
codecs="mp4a.40.2">
        <AudioChannelConfiguration
schemeIdUri="urn:mpeg:dash:23003:3:audio_channel_configuration:2011" value="2"/>
    </Representation>
    <Representation id="4" bandwidth="193007" audioSamplingRate="48000"
codecs="mp4a.40.2">
        <AudioChannelConfiguration
schemeIdUri="urn:mpeg:dash:23003:3:audio_channel_configuration:2011" value="2"/>
    </Representation>
  </AdaptationSet>
</Period>

```

```

    <Representation id="6" bandwidth="193007" audioSamplingRate="48000"
codecs="mp4a.40.2">
      <AudioChannelConfiguration
schemeIdUri="urn:mpeg:dash:23003:3:audio_channel_configuration:2011" value="2"/>
    </Representation>
  </AdaptationSet>
  <SupplementalProperty schemeIdUri="urn:scte:dash:utc-time"
value="2023-02-10T21:02:31.007Z"/>
</Period>
<Period start="PT80445.560S" id="155" duration="PT44.978S">
  <EventStream timescale="90000" schemeIdUri="urn:scte:scte35:2013:xml">
    <Event duration="4048044">
      <scte35:SpliceInfoSection protocolVersion="0" ptsAdjustment="207000"
tier="4095">
        <scte35:SpliceInsert spliceEventId="111" spliceEventCancelIndicator="false"
outOfNetworkIndicator="true" spliceImmediateFlag="false" uniqueProgramId="1"
availNum="1" availsExpected="1">
          <scte35:Program>
            <scte35:SpliceTime ptsTime="7239893422"/>
          </scte35:Program>
          <scte35:BreakDuration autoReturn="true" duration="4048044"/>
        </scte35:SpliceInsert>
      </scte35:SpliceInfoSection>
    </Event>
  </EventStream>
  <AdaptationSet id="1485523442" mimeType="video/mp4" segmentAlignment="true"
startWithSAP="1" subsegmentAlignment="true" subsegmentStartsWithSAP="1"
bitstreamSwitching="true">
    <SegmentTemplate timescale="60000" media="index_video_${RepresentationID$_0_
$Number$.mp4?m=1676062374" initialization="index_video_${RepresentationID$_0_init.mp4?
m=1676062374" startNumber="156" presentationTimeOffset="4826733614">
      <SegmentTimeline>
        <S t="4826733614" d="284284"/>
        <S t="4827017898" d="360360" r="5"/>
        <S t="4829180058" d="252252"/>
      </SegmentTimeline>
    </SegmentTemplate>
    <Representation id="1" width="960" height="540" frameRate="30000/1001"
bandwidth="1800000" codecs="avc1.4D401F"/>
    <Representation id="3" width="640" height="360" frameRate="30000/1001"
bandwidth="1200000" codecs="avc1.4D401E"/>
    <Representation id="5" width="480" height="270" frameRate="30000/1001"
bandwidth="800000" codecs="avc1.4D4015"/>
  </AdaptationSet>

```



```

    <AdaptationSet id="1377232898" mimeType="audio/mp4" segmentAlignment="0"
    lang="eng">
      <Label>eng</Label>
      <SegmentTemplate timescale="48000" media="index_audio_$RepresentationID$_
$Number$.mp4?m=1676062374" initialization="index_audio_$RepresentationID$_init.mp4?
m=1676062374" startNumber="156" presentationTimeOffset="3861387246">
        <SegmentTimeline>
          <S t="3861387246" d="227328"/>
          <S t="3861614574" d="288768"/>
          <S t="3861903342" d="287744"/>
          <S t="3862191086" d="288768"/>
          <S t="3862479854" d="287744"/>
          <S t="3862767598" d="288768"/>
          <S t="3863056366" d="287744"/>
          <S t="3863344110" d="202752"/>
        </SegmentTimeline>
      </SegmentTemplate>
      <Representation id="2" bandwidth="193007" audioSamplingRate="48000"
      codecs="mp4a.40.2">
        <AudioChannelConfiguration
        schemeIdUri="urn:mpeg:dash:23003:3:audio_channel_configuration:2011" value="2"/>
      </Representation>
      <Representation id="4" bandwidth="193007" audioSamplingRate="48000"
      codecs="mp4a.40.2">
        <AudioChannelConfiguration
        schemeIdUri="urn:mpeg:dash:23003:3:audio_channel_configuration:2011" value="2"/>
      </Representation>
      <Representation id="6" bandwidth="193007" audioSamplingRate="48000"
      codecs="mp4a.40.2">
        <AudioChannelConfiguration
        schemeIdUri="urn:mpeg:dash:23003:3:audio_channel_configuration:2011" value="2"/>
      </Representation>
    </AdaptationSet>
    <SupplementalProperty schemeIdUri="urn:scte:dash:utc-time"
    value="2023-02-10T21:07:35.111Z"/>
  </Period>
  <Period start="PT80490.538S" id="163">
    <AdaptationSet id="1485523442" mimeType="video/mp4" segmentAlignment="true"
    startWithSAP="1" subsegmentAlignment="true" subsegmentStartsWithSAP="1"
    bitstreamSwitching="true">
      <SegmentTemplate timescale="60000" media="index_video_$RepresentationID$_
$Number$.mp4?m=1676062374" initialization="index_video_$RepresentationID$_init.mp4?
m=1676062374" startNumber="164" presentationTimeOffset="4829432310">
        <SegmentTimeline>

```

```

        <S t="4829432310" d="348348"/>
        <S t="4829780658" d="360360" r="1"/>
    </SegmentTimeline>
</SegmentTemplate>
    <Representation id="1" width="960" height="540" frameRate="30000/1001"
bandwidth="1800000" codecs="avc1.4D401F"/>
    <Representation id="3" width="640" height="360" frameRate="30000/1001"
bandwidth="1200000" codecs="avc1.4D401E"/>
    <Representation id="5" width="480" height="270" frameRate="30000/1001"
bandwidth="800000" codecs="avc1.4D4015"/>
</AdaptationSet>
    <AdaptationSet id="1377232898" mimeType="audio/mp4" segmentAlignment="0"
lang="eng">
        <Label>eng</Label>
        <SegmentTemplate timescale="48000" media="index_audio_$RepresentationID$_0_
$Number$.mp4?m=1676062374" initialization="index_audio_$RepresentationID$_0_init.mp4?
m=1676062374" startNumber="164" presentationTimeOffset="3863546862">
            <SegmentTimeline>
                <S t="3863546862" d="278528"/>
                <S t="3863825390" d="287744"/>
                <S t="3864113134" d="288768"/>
            </SegmentTimeline>
        </SegmentTemplate>
        <Representation id="2" bandwidth="193007" audioSamplingRate="48000"
codecs="mp4a.40.2">
            <AudioChannelConfiguration
schemeIdUri="urn:mpeg:dash:23003:3:audio_channel_configuration:2011" value="2"/>
        </Representation>
        <Representation id="4" bandwidth="193007" audioSamplingRate="48000"
codecs="mp4a.40.2">
            <AudioChannelConfiguration
schemeIdUri="urn:mpeg:dash:23003:3:audio_channel_configuration:2011" value="2"/>
        </Representation>
        <Representation id="6" bandwidth="193007" audioSamplingRate="48000"
codecs="mp4a.40.2">
            <AudioChannelConfiguration
schemeIdUri="urn:mpeg:dash:23003:3:audio_channel_configuration:2011" value="2"/>
        </Representation>
    </AdaptationSet>
    <SupplementalProperty schemeIdUri="urn:scte:dash:utc-time"
value="2023-02-10T21:08:20.090Z"/>
</Period>
</MPD>

```

Example Linear DASH personalized manifest (with creative ad signaling):

```

<MPD availabilityStartTime="2023-02-09T22:47:05.865000+00:00"
  id="201" minBufferTime="PT10S" minimumUpdatePeriod="PT6S"
  profiles="urn:mpeg:dash:profile:isoff-live:2011"
  publishTime="2023-02-10T21:08:43+00:00" suggestedPresentationDelay="PT20.000S"
  timeShiftBufferDepth="PT88.999S" type="dynamic" xmlns="urn:mpeg:dash:schema:mpd:2011"
  xmlns:scte35="urn:scte:scte35:2013:xml" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:schemaLocation="urn:mpeg:dash:schema:mpd:2011 http://standards.iso.org/
ittf/PubliclyAvailableStandards/MPEG-DASH_schema_files/DASH-MPD.xsd">
  <BaseURL>https://d3fch9e2fcarly.cloudfront.net/out/v1/
f9f38deca3f14fc4b5ab3cdbd76cfb9e/</BaseURL>
  <Location>https://777788889999.mediatailor.us-west-2.amazonaws.com/
v1/dash/94063eadf7d8c56e9e2edd84fdf897826a70d0df/emt/out/v1/
f9f38deca3f14fc4b5ab3cdbd76cfb9e/index.mpd?
aws.sessionId=672ed481-4ffd-4270-936f-7c8403947f2e</Location>
  <Period duration="PT304.103S" id="104" start="PT80141.456S">
    <AdaptationSet bitstreamSwitching="true" id="1485523442" mimeType="video/
mp4" segmentAlignment="true" startWithSAP="1" subsegmentAlignment="true"
subsegmentStartsWithSAP="1">
      <SegmentTemplate initialization="index_video_$RepresentationID$_0_init.mp4?
m=1676062374" media="index_video_$RepresentationID$_0_$Number$.mp4?m=1676062374"
presentationTimeOffset="4808487386" startNumber="151" timescale="60000">
        <SegmentTimeline>
          <S d="360360" r="3" t="4824975858"/>
          <S d="316316" t="4826417298"/>
        </SegmentTimeline>
      </SegmentTemplate>
      <Representation bandwidth="1800000" codecs="avc1.4D401F"
frameRate="30000/1001" height="540" id="1" width="960"/>
      <Representation bandwidth="1200000" codecs="avc1.4D401E"
frameRate="30000/1001" height="360" id="3" width="640"/>
      <Representation bandwidth="800000" codecs="avc1.4D4015"
frameRate="30000/1001" height="270" id="5" width="480"/>
    </AdaptationSet>
    <AdaptationSet id="1377232898" lang="eng" mimeType="audio/mp4"
segmentAlignment="0">
      <Label>eng</Label>
      <SegmentTemplate initialization="index_audio_$RepresentationID$_0_init.mp4?
m=1676062374" media="index_audio_$RepresentationID$_0_$Number$.mp4?m=1676062374"
presentationTimeOffset="3846790126" startNumber="151" timescale="48000">
        <SegmentTimeline>
          <S d="287744" t="3859981294"/>
          <S d="288768" t="3860269038"/>
        </SegmentTimeline>
      </SegmentTemplate>
    </AdaptationSet>
  </Period>
</MPD>

```

```

        <S d="287744" t="3860557806"/>
        <S d="288768" t="3860845550"/>
        <S d="252928" t="3861134318"/>
    </SegmentTimeline>
</SegmentTemplate>
<Representation audioSamplingRate="48000" bandwidth="193007"
codecs="mp4a.40.2" id="2">
    <AudioChannelConfiguration
schemeIdUri="urn:mpeg:dash:23003:3:audio_channel_configuration:2011" value="2"/>
</Representation>
<Representation audioSamplingRate="48000" bandwidth="193007"
codecs="mp4a.40.2" id="4">
    <AudioChannelConfiguration
schemeIdUri="urn:mpeg:dash:23003:3:audio_channel_configuration:2011" value="2"/>
</Representation>
<Representation audioSamplingRate="48000" bandwidth="193007"
codecs="mp4a.40.2" id="6">
    <AudioChannelConfiguration
schemeIdUri="urn:mpeg:dash:23003:3:audio_channel_configuration:2011" value="2"/>
</Representation>
</AdaptationSet>
<SupplementalProperty schemeIdUri="urn:scte:dash:utc-time"
value="2023-02-10T21:02:31.007Z"/>
</Period>
<Period id="155_1" start="PT22H20M45.56S">
    <BaseURL>https://777788889999.mediatailor.us-west-2.amazonaws.com/
v1/dashsegment/94063eadf7d8c56e9e2edd84fdf897826a70d0df/
emt/672ed481-4ffd-4270-936f-7c8403947f2e/155/155_1/</BaseURL>
    <EventStream schemeIdUri="urn:sva:advertising-wg:ad-id-signaling"
timescale="90000">
        <Event presentationTime="xxxxx" duration="1351350">
            <![CDATA[{"version": 1,"identifiers": [{"scheme":
"urn:smp:ul:060E2B34.01040101.01200900.00000000","value": "155_1","ad_position":
"155_1", "ad_type":"avail","creative_id": "123","tracking_uri": "../..../v1/
tracking/hashed-account-id/origin-id/session-id","custom_vast_data":"123abc"}]]]>
        </Event>
    </EventStream>
    <AdaptationSet bitstreamSwitching="false" frameRate="30000/1001"
mimeType="video/mp4" segmentAlignment="true" startWithSAP="1"
subsegmentAlignment="true" subsegmentStartsWithSAP="1">
        <SegmentTemplate startNumber="1" timescale="90000"/>
        <Representation bandwidth="1800000" codecs="avc1.64001f" height="540"
id="1" width="960">

```

```

        <SegmentTemplate initialization="asset_540_1_2init.mp4"
media="asset_540_1_2_${Number%09d$.mp4" startNumber="1" timescale="90000">
            <SegmentTimeline>
                <S d="180180" r="6" t="0"/>
                <S d="90090" t="1261260"/>
            </SegmentTimeline>
        </SegmentTemplate>
    </Representation>
    <Representation bandwidth="1200000" codecs="avc1.64001e" height="360"
id="3" width="640">
        <SegmentTemplate initialization="asset_360_1_1init.mp4"
media="asset_360_1_1_${Number%09d$.mp4" startNumber="1" timescale="90000">
            <SegmentTimeline>
                <S d="180180" r="6" t="0"/>
                <S d="90090" t="1261260"/>
            </SegmentTimeline>
        </SegmentTemplate>
    </Representation>
    <Representation bandwidth="800000" codecs="avc1.640015" height="270" id="5"
width="480">
        <SegmentTemplate initialization="asset_270_0_0init.mp4"
media="asset_270_0_0_${Number%09d$.mp4" startNumber="1" timescale="90000">
            <SegmentTimeline>
                <S d="180180" r="6" t="0"/>
                <S d="90090" t="1261260"/>
            </SegmentTimeline>
        </SegmentTemplate>
    </Representation>
</AdaptationSet>
<AdaptationSet lang="eng" mimeType="audio/mp4" segmentAlignment="0">
    <SegmentTemplate initialization="asset_audio_128_3init.mp4"
media="asset_audio_128_3_${Number%09d$.mp4" startNumber="1" timescale="48000"/>
    <Label>eng</Label>
    <Representation audioSamplingRate="48000" bandwidth="128000"
codecs="mp4a.40.2" id="6">
        <SegmentTemplate initialization="asset_audio_128_3init.mp4"
media="asset_audio_128_3_${Number%09d$.mp4" startNumber="1" timescale="48000">
            <SegmentTimeline>
                <S d="98304" t="0"/>
                <S d="96256" r="1" t="98304"/>
                <S d="95232" t="290816"/>
                <S d="96256" r="2" t="386048"/>
                <S d="48128" t="674816"/>
            </SegmentTimeline>

```

```

        </SegmentTemplate>
        <AudioChannelConfiguration
schemeIdUri="urn:mpeg:dash:23003:3:audio_channel_configuration:2011" value="2"/>
        </Representation>
    </AdaptationSet>
</Period>
<Period id="155_2" start="PT22H21M0.575S">
    <BaseURL>https://777788889999.mediatailor.us-west-2.amazonaws.com/
v1/dashsegment/94063eadf7d8c56e9e2edd84fdf897826a70d0df/
emt/672ed481-4ffd-4270-936f-7c8403947f2e/155/155_2/</BaseURL>
    <EventStream schemeIdUri="urn:sva:advertising-wg:ad-id-signaling"
timescale="90000">
        <Event presentationTime="0" duration="1351350">
            <![CDATA[{"version": 1,"identifiers": [{"scheme":
"urn:smp:ul:060E2B34.01040101.01200900.00000000","value": "155_2","ad_position":
"155_2", "ad_type":"avail","creative_id": "234","tracking_uri": "../.../v1/
tracking/hashed-account-id/origin-id/session-id","custom_vast_data":"123abc"}]]]>
        </Event>
    </EventStream>
    <AdaptationSet bitstreamSwitching="false" frameRate="30000/1001"
mimeType="video/mp4" segmentAlignment="true" startWithSAP="1"
subsegmentAlignment="true" subsegmentStartsWithSAP="1">
        <SegmentTemplate startNumber="1" timescale="90000"/>
        <Representation bandwidth="1800000" codecs="avc1.64001f" height="540"
id="1" width="960">
            <SegmentTemplate initialization="asset_540_1_2init.mp4"
media="asset_540_1_2_${Number%09d$.mp4" startNumber="1" timescale="90000">
                <SegmentTimeline>
                    <S d="180180" r="6" t="0"/>
                    <S d="90090" t="1261260"/>
                </SegmentTimeline>
            </SegmentTemplate>
        </Representation>
        <Representation bandwidth="1200000" codecs="avc1.64001e" height="360"
id="3" width="640">
            <SegmentTemplate initialization="asset_360_1_1init.mp4"
media="asset_360_1_1_${Number%09d$.mp4" startNumber="1" timescale="90000">
                <SegmentTimeline>
                    <S d="180180" r="6" t="0"/>
                    <S d="90090" t="1261260"/>
                </SegmentTimeline>
            </SegmentTemplate>
        </Representation>
    </Period>
</Manifest>

```

```

        <Representation bandwidth="800000" codecs="avc1.640015" height="270" id="5"
width="480">
            <SegmentTemplate initialization="asset_270_0_0init.mp4"
media="asset_270_0_0_$.Number%09d$.mp4" startNumber="1" timescale="90000">
                <SegmentTimeline>
                    <S d="180180" r="6" t="0"/>
                    <S d="90090" t="1261260"/>
                </SegmentTimeline>
            </SegmentTemplate>
        </Representation>
    </AdaptationSet>
    <AdaptationSet lang="eng" mimeType="audio/mp4" segmentAlignment="0">
        <SegmentTemplate initialization="asset_audio_128_3init.mp4"
media="asset_audio_128_3_$.Number%09d$.mp4" startNumber="1" timescale="48000"/>
        <Label>eng</Label>
        <Representation audioSamplingRate="48000" bandwidth="128000"
codecs="mp4a.40.2" id="6">
            <SegmentTemplate initialization="asset_audio_128_3init.mp4"
media="asset_audio_128_3_$.Number%09d$.mp4" startNumber="1" timescale="48000">
                <SegmentTimeline>
                    <S d="98304" t="0"/>
                    <S d="96256" r="1" t="98304"/>
                    <S d="95232" t="290816"/>
                    <S d="96256" r="2" t="386048"/>
                    <S d="48128" t="674816"/>
                </SegmentTimeline>
            </SegmentTemplate>
            <AudioChannelConfiguration
schemeIdUri="urn:mpeg:dash:23003:3:audio_channel_configuration:2011" value="2"/>
        </Representation>
    </AdaptationSet>
</Period>
<Period id="155_3" start="PT22H21M15.59S">
    <BaseURL>https://777788889999.mediatailor.us-west-2.amazonaws.com/
v1/dashsegment/94063eadf7d8c56e9e2edd84fdf897826a70d0df/
emt/672ed481-4ffd-4270-936f-7c8403947f2e/155/155_3/</BaseURL>
    <EventStream schemeIdUri="urn:sva:advertising-wg:ad-id-signaling"
timescale="90000">
        <Event presentationTime="0" duration="1351350">
            <![CDATA[{"version": 1,"identifiers": [{"scheme":
"urn:smp:ul:060E2B34.01040101.01200900.00000000","value": "155_3","ad_position":
"155_3", "ad_type":"avail","creative_id": "345","tracking_uri": "../v1/
tracking/hashed-account-id/origin-id/session-id","custom_vast_data":"123abc"}]]]>
        </Event>
    </EventStream>
</Period>

```

```

    </EventStream>
    <AdaptationSet bitstreamSwitching="false" frameRate="30000/1001"
    mimeType="video/mp4" segmentAlignment="true" startWithSAP="1"
    subsegmentAlignment="true" subsegmentStartsWithSAP="1">
        <SegmentTemplate startNumber="1" timescale="90000"/>
        <Representation bandwidth="1800000" codecs="avc1.64001f" height="540"
    id="1" width="960">
            <SegmentTemplate initialization="asset_540_1_2init.mp4"
    media="asset_540_1_2_${Number%09d$.mp4" startNumber="1" timescale="90000">
                <SegmentTimeline>
                    <S d="180180" r="6" t="0"/>
                    <S d="90090" t="1261260"/>
                </SegmentTimeline>
            </SegmentTemplate>
        </Representation>
        <Representation bandwidth="1200000" codecs="avc1.64001e" height="360"
    id="3" width="640">
            <SegmentTemplate initialization="asset_360_1_1init.mp4"
    media="asset_360_1_1_${Number%09d$.mp4" startNumber="1" timescale="90000">
                <SegmentTimeline>
                    <S d="180180" r="6" t="0"/>
                    <S d="90090" t="1261260"/>
                </SegmentTimeline>
            </SegmentTemplate>
        </Representation>
        <Representation bandwidth="800000" codecs="avc1.640015" height="270" id="5"
    width="480">
            <SegmentTemplate initialization="asset_270_0_0init.mp4"
    media="asset_270_0_0_${Number%09d$.mp4" startNumber="1" timescale="90000">
                <SegmentTimeline>
                    <S d="180180" r="6" t="0"/>
                    <S d="90090" t="1261260"/>
                </SegmentTimeline>
            </SegmentTemplate>
        </Representation>
    </AdaptationSet>
    <AdaptationSet lang="eng" mimeType="audio/mp4" segmentAlignment="0">
        <SegmentTemplate initialization="asset_audio_128_3init.mp4"
    media="asset_audio_128_3_${Number%09d$.mp4" startNumber="1" timescale="48000"/>
        <Label>eng</Label>
        <Representation audioSamplingRate="48000" bandwidth="128000"
    codecs="mp4a.40.2" id="6">
            <SegmentTemplate initialization="asset_audio_128_3init.mp4"
    media="asset_audio_128_3_${Number%09d$.mp4" startNumber="1" timescale="48000">

```



```

        <SegmentTimeline>
            <S d="98304" t="0"/>
            <S d="96256" r="1" t="98304"/>
            <S d="95232" t="290816"/>
            <S d="96256" r="2" t="386048"/>
            <S d="48128" t="674816"/>
        </SegmentTimeline>
    </SegmentTemplate>
    <AudioChannelConfiguration
schemeIdUri="urn:mpeg:dash:23003:3:audio_channel_configuration:2011" value="2"/>
    </Representation>
</AdaptationSet>
</Period>
<Period id="163" start="PT80490.538S">
    <AdaptationSet bitstreamSwitching="true" id="1485523442" mimeType="video/
mp4" segmentAlignment="true" startWithSAP="1" subsegmentAlignment="true"
subsegmentStartsWithSAP="1">
        <SegmentTemplate initialization="index_video_$RepresentationID$_0_init.mp4?
m=1676062374" media="index_video_$RepresentationID$_0_$Number$.mp4?m=1676062374"
presentationTimeOffset="4829432310" startNumber="164" timescale="60000">
            <SegmentTimeline>
                <S d="348348" t="4829432310"/>
                <S d="360360" r="1" t="4829780658"/>
            </SegmentTimeline>
        </SegmentTemplate>
        <Representation bandwidth="1800000" codecs="avc1.4D401F"
frameRate="30000/1001" height="540" id="1" width="960"/>
        <Representation bandwidth="1200000" codecs="avc1.4D401E"
frameRate="30000/1001" height="360" id="3" width="640"/>
        <Representation bandwidth="800000" codecs="avc1.4D4015"
frameRate="30000/1001" height="270" id="5" width="480"/>
    </AdaptationSet>
    <AdaptationSet id="1377232898" lang="eng" mimeType="audio/mp4"
segmentAlignment="0">
        <Label>eng</Label>
        <SegmentTemplate initialization="index_audio_$RepresentationID$_0_init.mp4?
m=1676062374" media="index_audio_$RepresentationID$_0_$Number$.mp4?m=1676062374"
presentationTimeOffset="3863546862" startNumber="164" timescale="48000">
            <SegmentTimeline>
                <S d="278528" t="3863546862"/>
                <S d="287744" t="3863825390"/>
                <S d="288768" t="3864113134"/>
            </SegmentTimeline>
        </SegmentTemplate>
    </AdaptationSet>
</Period>
</Manifest>

```

```

        <Representation audioSamplingRate="48000" bandwidth="193007"
codecs="mp4a.40.2" id="2">
            <AudioChannelConfiguration
schemeIdUri="urn:mpeg:dash:23003:3:audio_channel_configuration:2011" value="2"/>
        </Representation>
        <Representation audioSamplingRate="48000" bandwidth="193007"
codecs="mp4a.40.2" id="4">
            <AudioChannelConfiguration
schemeIdUri="urn:mpeg:dash:23003:3:audio_channel_configuration:2011" value="2"/>
        </Representation>
        <Representation audioSamplingRate="48000" bandwidth="193007"
codecs="mp4a.40.2" id="6">
            <AudioChannelConfiguration
schemeIdUri="urn:mpeg:dash:23003:3:audio_channel_configuration:2011" value="2"/>
        </Representation>
    </AdaptationSet>
    <SupplementalProperty schemeIdUri="urn:scte:dash:utc-time"
value="2023-02-10T21:08:20.090Z"/>
</Period>
</MPD>

```

Ad Decision Server (ADS) interactions

MediaTailor uses the creative id attribute value from the VAST response as a value in the ad ID signaling. If the id attribute value is empty or not present in the VAST response, MediaTailor places an empty value in the ad ID signaling.

Example VAST response:

The following sample VAST response includes an id attribute value for the inline linear Creative. MediaTailor extracts the value from the custom VAST Extension element and places that value in the creative metadata of the manifest.

```

<?xml version="1.0" encoding="utf-8"?>
<VAST version="3.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <Ad sequence="3">
        <Inline>
            <AdSystem>2.0</AdSystem>
            <AdTitle>AD-caribbean2-15</AdTitle>
            <Impression><![CDATA[https://n8ljfs0xxx.execute-api.us-
west-2.amazonaws.com/v1/impression]]></Impression>
            <Creatives>

```

```

        <Creative sequence="3" apiFramework="inLine" id="1234">
            <Linear>
                <Duration>00:00:15</Duration>
                <MediaFiles>
                    <MediaFile id="00002" delivery="progressive" type="video/
mp4" width="1280" height="720"><![CDATA[https://d3re4i3vgppxxx.cloudfront.net/Media/
Bumpers/AD-caribbean2-15-HD.mp4]]></MediaFile>
                </MediaFiles>
            </Linear>
        </Creative>
    </Creatives>
    <Extensions>
        <Extension type="creative_signaling"><![CDATA[999999|
TVN1DDNpFTchtpRj,E5TfTtcYd5IEzvEt,ChA050HcvWRGFY6Zp5VSS1xUEJ2B9p8GGhQIDzIQkFeQC-
Ho67FR3P9qNa6khSAGKgAyAA]]></Extension>
    </Extensions>
</InLine>
</Ad>
</VAST>

```

Client-side tracking API

The following example shows how a player SDK links the ad metadata in the manifest with the full tracking event data in the client-side tracking response payload with `creativeId` and `adId`.

Example JSON message:

```

{
  "avails": [
    {
      "adBreakTrackingEvents": [],
      "ads": [
        {
          "adId": "5",
          "adParameters": "",
          "adProgramDateTime": null,
          "adSystem": "2.0",
          "adTitle": "AD-caribbean2-15",
          "adVerifications": [],
          "companionAds": [],
          "creativeId": "1234",
          "creativeSequence": "2",
          "duration": "PT15S",

```

```

    "durationInSeconds": 15,
    "extensions": [],
    "mediaFiles": {
      "mediaFilesList": [],
      "mezzanine": ""
    },
    "skipOffset": null,
    "startTime": "PT30S",
    "startTimeInSeconds": 30,
    "trackingEvents": [
      {
        "beaconUrls": [
          "https://myServer/impression"
        ],
        "duration": "PT15S",
        "durationInSeconds": 15,
        "eventId": "5",
        "eventProgramDateTime": null,
        "eventType": "impression",
        "startTime": "PT30S",
        "startTimeInSeconds": 30
      }
    ],
    "vastAdId": ""
  }
],
"availId": "5",
"availProgramDateTime": null,
"duration": "PT15S",
"durationInSeconds": 15,
"meta": null,
"nonLinearAdsList": [],
"startTime": "PT30S",
"startTimeInSeconds": 30
}
],
"nextToken": "UFQ1TTM0Ljk2N1NfMjAyMi0xMS0xOFQwNDozMzo1Mi4yNDUxOTdaXzE%3D",
"nonLinearAvails": []
}

```

Using AWS Elemental MediaTailor to create linear assembled streams

AWS Elemental MediaTailor channel assembly is a manifest-only service that allows you to create linear streaming channels using your existing video on demand (VOD) content mixed with live content. MediaTailor never touches your content segments, which are served directly from your origin server. Instead, MediaTailor fetches the manifests from your origin, and uses them to assemble a live sliding manifest window that references the underlying content segments. Channel assembly keeps track of things like the media sequence number that's necessary to make playback smooth from asset to asset. Linear assembled streams are created with a low running cost by using existing multi-bitrate encoded and packaged VOD content.

You can easily monetize channel assembly linear streams by inserting ad breaks in your programs without having to condition the content with SCTE-35 markers. You can use channel assembly with the MediaTailor ad insertion service, or any server-side ad insertion service.

To get started with channel assembly, see [the section called "Getting started with MediaTailor channel assembly"](#).

For information about integrating Channel Assembly with a content delivery network (CDN), see [Build MediaTailor linear channels with channel assembly and CDN](#).

Topics

- [Channel Assembly workflow types](#)
- [Working with source locations](#)
- [Working with channels](#)
- [Adding a program to a channel's schedule](#)
- [Insert personalized ads and ad breaks in a channel stream](#)
- [Time-shifting a channel's playback](#)

Channel Assembly workflow types

AWS Elemental MediaTailor Channel Assembly supports three workflow types for creating linear streaming channels. Each workflow type uses a different combination of source content to meet

specific use cases. This topic describes each workflow type, when to use it, and links to the detailed setup procedures.

Workflow type	Description	Playback mode	Use case
VOD-to-live	Assembles pre-recorded VOD assets into a live linear stream	Loop or Linear	24/7 channels from a content library
Live-to-live	Passes through live source feeds as programs in a scheduled channel	Linear only	News, sports, regional programming
VOD + Live-to-live (Mixed)	Combines VOD and live sources in a single channel schedule	Linear only	Channels with mostly VOD content that include scheduled live events

VOD-to-live

A VOD-to-live channel creates a linear stream entirely from pre-encoded, pre-packaged VOD content. The stream looks like a live broadcast with a live sliding manifest window, but uses only VOD assets. You don't need SCTE-35 markers in your source content because you define ad breaks when you schedule programs.

When to use

- You have a library of VOD assets and want to create a 24/7 linear channel.
- You want to monetize content with ad breaks but don't have SCTE-35 markers in your source content.
- You want a simple, low-cost linear stream that doesn't require live encoding infrastructure.

How it works

1. **Prepare your content** – Encode and package your VOD assets. Host them on an origin server (Amazon S3, AWS Elemental MediaPackage, or any HTTP origin).
2. **Create a source location** – Register your origin server with MediaTailor.

3. **Add VOD sources** – Register each piece of content and define its package configurations and source groups.
4. **Create a channel** – Choose a playback mode (Loop or Linear) and configure outputs.
5. **Schedule programs** – Add programs to your channel's schedule in the order you want them to play.
6. **Configure ad breaks** – Add ad breaks to each program. Connect to an SSAI configuration for personalized ads.
7. **Start your channel and configure a CDN** – Start the channel. Optionally, add a CDN in front of it for scale and cost savings.

Choosing a playback mode

VOD-to-live channels support two playback modes:

- **Loop (Basic tier)** – Programs play back-to-back in an endless loop. When the last program finishes, playback returns to the first. This is the simplest option for a 24/7 channel. Loop mode does not support live sources or filler slate.
- **Linear (Standard tier)** – Programs play once in sequence. Linear mode supports time-based scheduling and filler slate for gaps between programs.

Content requirements

For smooth transitions, all VOD assets in the same channel must have:

- The same number of renditions (video, audio, subtitle, and so on) across all sources in a source group
- Consistent segment durations across all assets. We recommend 2 to 6 seconds.
- Matching bitrates and resolutions for corresponding renditions

Important

Channel Assembly does not support per-title encoding or automated ABR encoding. These methods produce assets with different rendition counts and durations, which causes playback failures at program transitions.

Ad breaks and monetization

You define ad breaks when you add programs to your schedule. Each ad break needs an offset in milliseconds from the start of the program. The offset must align with a segment boundary on all tracks in the VOD source.

Your VOD source manifest might contain ad markers. For HLS, these are EXT-X-DATERANGE or EXT-X-CUE-OUT tags with a duration. For DASH, use EventStream. When MediaTailor detects these markers, it shows their offsets as options when you set up the ad break. This provides the most precise ad placement.

To serve personalized ads, pair your channel with a MediaTailor SSAI playback configuration. Set the content source to your channel's playback URL.

Tips

Topic	Detail
Ad break placement	Ad break offsets that don't align with a segment boundary are adjusted to the nearest boundary. Use shorter segments (2–4 seconds) for more precise ad placement.
Signed URL origins	MediaTailor caches VOD source manifests when you first add the source. If you use signed URLs with short expiration times, the cached URLs might expire before playback. Call the <code>UpdateVodSource</code> API to force MediaTailor to re-fetch the manifest.
Rendition mismatch	If a program fails to transition cleanly, check that all VOD sources in the channel have the same rendition counts and use the same ABR ladder.
Scheduling ahead	Add programs to a running Linear channel at least 10 minutes before their air time. Channel Assembly begins processing programs within this window.

For instructions on creating your first channel, see [the section called “Getting started with MediaTailor channel assembly”](#).

Live-to-live

A live-to-live channel passes through live source feeds as programs in a scheduled channel. The channel switches between different live sources based on your schedule. Use this workflow when all of your content is live and you want to assemble multiple live feeds into a single linear channel.

When to use

- You have multiple live encoders/packagegers and want to switch between them on a schedule.
- You're building a regional channel that switches between national and regional live feeds at different times of day.
- You have 24/7 live feeds (news, sports networks) and want to create a curated channel that draws from multiple sources.

How it works

1. **Set up your live encoding pipeline** – Encode and package your live content using AWS Elemental MediaLive, AWS Elemental MediaPackage, or any live packaging origin.
2. **Create a source location** – Register the origin server for your live content with MediaTailor.
3. **Add live sources** – Register each live feed and define its package configurations and source groups.
4. **Create a channel** – Choose the Linear playback mode (required for live sources) and configure outputs.
5. **Schedule programs** – Add live source programs to your channel's schedule. Specify the duration for each live program.
6. **Start your channel and configure a CDN** – Start the channel. Optionally, add a CDN in front of it for scale and cost savings.

Playback mode

Live-to-live channels require the **Linear (Standard tier)** playback mode. Loop mode does not support live sources.

Content requirements

In addition to the general requirements listed in [the section called “Working with source locations”](#):

- **HLS live sources** – You must provide #EXT-X-PROGRAM-DATE-TIME tags for the first segment in the manifest window and on every discontinuity.
- **HLS ad markers** – You must configure ad markers as DATERANGE.
- **Source manifest window** – Use a manifest window duration that's at least 30 seconds longer than the manifest window on your Channel Assembly channel.
- **Target duration** – Make the target duration match the duration of the existing sources in the channel.
- **Child playlists** – Make the number of child playlists match that of the other sources in the channel.

Live source configuration with MediaPackage

If you use AWS Elemental MediaPackage as your packaging origin, we recommend the following configuration:

HLS settings:

Setting	Value	Necessity	Notes
Endpoint type	Apple HLS or CMAF	Required	Use Apple HLS for TS segments, CMAF for MP4 segments
ProgramDateTimeIntervalSeconds	1	Required	Prevents playback issues at discontinuities.
PlaylistWindowSeconds	30 seconds longer than the CA manifest window	Required	
AdMarkers	DATERANGE	Required	Required when passing through ad markers.

Setting	Value	Necessity	Notes
IncludeIframeOnlyStream	Disabled	Recommended	

DASH settings:

Setting	Value	Necessity	Notes
ManifestLayout	FULL	Recommended	
SegmentTemplateFormat	NUMBER_WITH_TIMELINE or TIME_WITH_TIMELINE	Recommended	NUMBER_WITH_DURATION is not supported.
ManifestWindowSeconds	30 seconds longer than the CA manifest window	Required	
PeriodTriggers	ADS	Required	Required when passing through ad markers.

Ad breaks

For live sources, you don't define ad breaks in the Channel Assembly schedule. Instead, ad breaks in the live source manifests are passed through automatically:

- **HLS** – Ad breaks using the EXT-X-DATERANGE tag are passed through.
- **DASH** – Ad breaks using EventStream are passed through.

To monetize passed-through ad breaks, pair your channel with a MediaTailor SSAI playback configuration.

Tips

Topic	Detail
Program duration	You must specify a duration for each live source program. When the duration expires, Channel Assembly transitions to the next program in the schedule.
Filler slate	If there are gaps between programs in the schedule, MediaTailor plays filler slate. If the duration of the slate is less than the duration of the gap, MediaTailor loops the slate. Configure filler slate when you create your channel.
Regional channels	Use live sources to create regional channels. Run one encoder/packager for national content, then run regional encoders when those regions are live. Create a separate channel per region, each with its own schedule.
Transition timing	MediaTailor starts playback of a program on a common segment boundary. For absolute transitions, expect up to one segment of variance between the scheduled time and the actual transition.
Encoding costs	Minimize costs by running encoders only when content is live. Configure filler slate on your channel to cover gaps between live events.

VOD + Live-to-live (Mixed)

A mixed channel combines both VOD and live sources in a single schedule. This is the most flexible workflow type. Use it when your channel is primarily VOD content with scheduled live events interspersed, or when you want to bridge gaps between live events with pre-recorded content.

When to use

- Your channel is mostly VOD content, but you have scheduled live events (nightly news, weekly sports, special broadcasts).
- You want to switch between VOD filler content and live feeds throughout the day.

- You're building a channel that uses pre-recorded content between live event windows.

How it works

1. **Set up your content pipeline** – Encode and package both your VOD assets and live content. Host them on origin servers.
2. **Create source locations** – Register your origin servers with MediaTailor. You can use separate source locations for VOD and live content, or a single source location if they share an origin.
3. **Add VOD sources and live sources** – Register your VOD assets and live feeds, each with their package configurations and source groups.
4. **Create a channel** – Choose the Linear playback mode (required when using live sources) and configure outputs.
5. **Schedule programs** – Add a mix of VOD and live source programs to your channel's schedule. Use absolute transitions for live programs that must start at specific times. Use relative transitions for VOD programs that fill gaps.
6. **Configure ad breaks** – Add ad breaks to VOD programs. For live programs, ad breaks are passed through from the source manifest.
7. **Start your channel and configure a CDN** – Start the channel. Optionally, add a CDN in front of it for scale and cost savings.

Playback mode

Mixed channels require the **Linear (Standard tier)** playback mode. Loop mode does not support live sources.

Content requirements

Mixed channels must meet the content requirements for both VOD and live sources:

- All sources (VOD and live) within a source group must have the same number of renditions.
- VOD and live sources must have matching target segment durations.
- Live sources must meet the requirements for live sources, including EXT-X-PROGRAM-DATE-TIME tags for HLS.

Scheduling patterns

Mixed channels support two transition types for programs:

- **Relative** – The program plays before or after another program in the schedule. Available for VOD source programs only.
- **Absolute** – The program plays at a specific wall-clock time. Available for both VOD and live source programs in Linear mode.

Common scheduling patterns:

Pattern	Description	Example
VOD filler with scheduled live windows	VOD programs fill the schedule, with live programs inserted at specific times.	A movie channel that switches to a live sports event at 8:00 PM.
Live primary with VOD interstitials	A live feed runs most of the time, with VOD programs during breaks or off-hours.	A news channel that plays pre-recorded segments between live broadcasts.
Time-of-day switching	Different sources (live or VOD) play at different times of day.	National feed during prime time, regional live feed in the morning, VOD content overnight.

Ad breaks

Ad break behavior depends on the source type of each program:

- **VOD programs** – You define ad breaks in the schedule with an offset in milliseconds. Connect to an SSAI configuration for personalized ads.
- **Live programs** – Ad breaks are passed through from the live source manifest automatically. HLS uses EXT-X-DATERANGE tags; DASH uses EventStream.

Tips

Topic	Detail
Transition from VOD to live	When transitioning from a VOD program to a live program at an absolute time, MediaTailor truncates the VOD program at the closest common segment boundary to the live program's start time.
Transition from live to VOD	When a live program's duration expires, MediaTailor transitions to the next program in the schedule. If the next program is VOD, playback is seamless as long as content requirements are met.
Filler slate	Configure filler slate on your channel. MediaTailor plays slate during any gaps between programs. If the live source becomes unavailable mid-program, Channel Assembly continues to serve the last known segments until the program duration expires.
Mixed ad monetization	You can use a single SSAI configuration for the entire channel. MediaTailor handles both scheduled ad breaks (VOD programs) and passed-through ad breaks (live programs) in the same stream.
Schedule flexibility	Add VOD programs at least 10 minutes ahead of air time. For live programs using absolute transitions, schedule them at least 15 minutes ahead on a running channel.

Choosing the right workflow

Use the following decision guide to choose the right workflow type for your use case:

- If you have *only VOD content* and don't need time-based scheduling → Use **VOD-to-live** with Loop playback mode.
- If you have *only VOD content* and need time-based scheduling → Use **VOD-to-live** with Linear playback mode.
- If you have *only live content* → Use **Live-to-live**.
- If you have *both VOD and live content* → Use **VOD + Live-to-live (Mixed)**.

Feature comparison

Feature	VOD-to-live (Loop)	VOD-to-live (Linear)	Live-to-live	Mixed
VOD sources	Yes	Yes	No	Yes
Live sources	No	No	Yes	Yes
Absolute transitions	No	Yes	Yes	Yes
Relative transitions	Yes	Yes	No	Yes (VOD only)
Scheduled ad breaks	Yes	Yes	No	Yes (VOD only)
Pass-through ad breaks	No	No	Yes	Yes (Live only)
Filler slate	No	Yes	Yes	Yes
24/7 looping	Yes	No	No	No
Time-based scheduling	No	Yes	Yes	Yes
Channel tier	Basic	Standard	Standard	Standard

Next steps

- [the section called “Getting started with MediaTailor channel assembly”](#) – Step-by-step tutorial for creating your first channel
- [the section called “Working with source locations”](#) – Detailed procedures for managing source locations and content
- [the section called “Working with channels”](#) – Channel configuration options and playback modes
- [the section called “Adding a program”](#) – Scheduling programs with VOD and live sources
- [the section called “Insert ads and ad breaks”](#) – Monetization with SSAI
- [the section called “Channel assembly with CDN”](#) – CDN integration for scale and cost savings

- [the section called “Time-shifting a channel's playback”](#) – Enable catch-up and start-over functionality

Working with source locations

A source location represents the origin server where your source content is stored. A source location can be Amazon S3, a standard web server, a content delivery network (CDN) such as Amazon CloudFront, or a packaging origin such as AWS Elemental MediaPackage. MediaTailor retrieves your content manifests from the source location, and uses them to assemble your channel's linear stream.

This topic explains how to use the AWS Elemental MediaTailor console to create and delete source locations, and how to work with VOD sources.

Topics

- [Creating a source location](#)
- [Configuring authentication for your source location](#)
- [Working with VOD sources](#)
- [Working with live sources](#)
- [Using package configurations](#)
- [Manifest caching](#)

Creating a source location

The following procedure explains how to create a source location using the MediaTailor console. For information about how to create source locations using the MediaTailor API, see [CreateSourceLocation](#) in the *AWS Elemental MediaTailor API Reference*.

To create a source location

1. Open the MediaTailor console at <https://console.aws.amazon.com/mediatailor/>.
2. In the navigation pane, choose **Channel assembly** > **Source locations**.
3. On the navigation bar, choose **Create source location**.
4. Under **Source location configuration**, enter a name and the base URL of your origin server:
 - **Name:** An identifier for your source location, such as **my-origin**.

- **Base URL:** The protocol and base URL of the origin server your content is stored, such as **https://111111111111.cloudfront.net**. The URL must be in a standard HTTP URL format, prefixed with **http://** or **https://**.

Optionally select **Use SigV4 for Amazon S3 authentication** if your source location is an Amazon S3 bucket, and if you'd like to use AWS Signature Version 4 for Amazon S3 access authentication. For advanced information, see [Configuring authentication for your source location](#).

5.

Under **Access configuration**, optionally configure authentication for your source location:

- **Access type:** Select the authentication type that MediaTailor uses to access the content stored on the source location's origin.
- **SigV4 for Amazon S3** - MediaTailor uses Amazon Signature Version 4 (SigV4) to authorize request to your origin. For more information, see [the section called "Authenticating requests to Amazon S3 with SigV4"](#).
- **Secrets Manager access token authentication** - MediaTailor uses Secrets Manager and a AWS KMS customer managed key created, owned, and managed by you to facilitate access token authentication between MediaTailor and your origin. For information about how to configure **Secrets Manager access token authentication**, see [the section called "Working with AWS Secrets Manager access token authentication"](#).
- **Header name** - Specify a HTTP header name. MediaTailor uses the HTTP header to send the access token to your origin in content manifest requests. You can use any header name as long as it doesn't start with `x-amz-` or `x-amzn-`. If you're integrating with [MediaPackage CDN authorization](#), the header value should be `X-MediaPackage-CDNIdentifier`.
- **Secret string key** - The `SecretString` key that you specified in your Secrets Manager secret. For example, if your `SecretString` contains a key and value pair such as: `{"MyHeaderName": "11111111-2222-3333-4444-111122223333"}`, then `MyHeaderName` is the `SecretString` key you enter in this field.
- **Secret ARN** - The ARN of the secret that holds your access token. For a step-by-step guide, see [Step 2: Create an AWS Secrets Manager secret](#).

6. Under **Segment delivery server configuration**, optionally configure a server to deliver your content segments:

- **Use a default segment delivery server:** Enter the base URL of the server that is used to deliver your content segments, such as a CDN. Configure **Default segment host name** if you'd like to use a different server than the source location server to serve the content segments. For example, you can restrict access to the origin manifests from players by using a different CDN configuration for the **Base HTTP URL** (what MediaTailor uses to access the manifests) and the **Default Segment Base URL** (what players uses to access the content segments). If you don't enter a value, MediaTailor defaults to the source location server for segment delivery.
 - **Use named segment delivery servers:** If you have configured a default segment delivery server, you can also configure additional segment delivery servers. Each one must have a unique name and a base URL. The base URL can be a full HTTP URL, or it can be a relative path like `/some/path/`. The names are used to identify which server should be used when MediaTailor receives a request for content segments. If the request contains the header `X-MediaTailor-SegmentDeliveryConfigurationName` and the value of the header matches a name, the corresponding base URL will be used to serve the content. If the header is not included in the request, or if it does not match any names, then the default segment delivery server will be used.
7. Choose **Create source location**.
 8. To add more source locations, repeat steps 2-6.

Configuring authentication for your source location

Use **access configuration** to configure authentication for your source location. When access configuration is on, MediaTailor only retrieves source manifests from your origin if the request is authorized between MediaTailor and your origin. Access configuration is turned off by default.

MediaTailor supports the following authentication types:

- SigV4 for Amazon S3 authentication
- AWS Secrets Manager access token
- SigV4 for MediaPackage version 2 (v2) authentication

This chapter explains how to use SigV4 for Amazon S3, MediaPackage v2, and AWS Secrets Manager access tokens for source location authentication.

For more information, select the applicable topic.

Topics

- [Authenticating requests to Amazon S3 with SigV4](#)
- [Working with SigV4 for MediaPackage Version 2](#)
- [Working with AWS Secrets Manager access token authentication](#)

Authenticating requests to Amazon S3 with SigV4

Signature Version 4 (SigV4) for Amazon S3 is a signing protocol used to authenticate requests to Amazon S3 over HTTPS. When you use SigV4 for Amazon S3, MediaTailor includes a signed authorization header in the HTTPS request to the Amazon S3 bucket used as your origin. If the signed authorization header is valid, your origin fulfills the request. If it isn't valid, the request fails.

For general information about SigV4 for AWS Key Management Service, see the [Authenticating Requests \(AWS Signature Version 4\)](#) topic in the *Amazon S3 API reference*.

Note

MediaTailor always signs requests to these origins with SigV4.

Requirements

If you activate SigV4 for Amazon S3 authentication for your source location, you must meet these requirements:

- You must allow MediaTailor to access your Amazon S3 bucket by granting **mediataylor.amazonaws.com** principal access in IAM. For information about configuring access in IAM, see [Access management](#) in the *AWS Identity and Access Management User Guide*.
- The **mediataylor.amazonaws.com** service principal must have permissions to read all multivariant playlists referenced by the VOD source package configurations.
- The caller of the API must have **s3:GetObject** IAM permissions to read all multivariant playlists referenced by your MediaTailor VOD source package configurations.
- Your MediaTailor source location base URL must follow the Amazon S3 virtual hosted-style request URL format. For example, `https://bucket-name.s3.Region.amazonaws.com/key-name`. For information about Amazon S3 hosted virtual-style access, see [Virtual Hosted-Style Requests](#).

Working with SigV4 for MediaPackage Version 2

Signature Version 4 (SigV4) for MediaPackage v2 is a signing protocol used to authenticate requests to MediaPackage v2 over HTTP. When you use SigV4 for MediaPackage v2, MediaTailor includes a signed authorization header in the HTTP request to the MediaPackage v2 endpoint used as your origin. If the signed authorization header is valid, your origin fulfills the request. If it isn't valid, the request fails.

For general information about SigV4 for MediaPackage v2, see the [Authenticating Requests \(AWS Signature Version 4\)](#) topic in the *MediaPackage v2 API reference*.

Requirements

If you activate SigV4 for MediaPackage v2 authentication for your source location, you must meet these requirements:

- You must allow MediaTailor to access your MediaPackage v2 endpoint by granting **mediatailor.amazonaws.com** principal access in an Origin Access Policy on the endpoint.
- Your MediaTailor source location base URL must be a MediaPackage v2 endpoint.
- The caller of the API must have **mediapackagev2:GetObject** IAM permissions to read all multivariant playlists referenced by the MediaTailor source packaging configurations.

Working with AWS Secrets Manager access token authentication

MediaTailor supports *Secrets Manager access token authentication*. With AWS Secrets Manager access token authentication, MediaTailor uses an AWS Key Management Service (AWS KMS) customer managed key and an AWS Secrets Manager secret that you create, own, and manage to authenticate requests to your origin.

In this section, we explain how Secrets Manager access token authentication works, and provide step-by-step information about how to configure Secrets Manager access token authentication. You can work with Secrets Manager access token authentication in the AWS Management Console or programmatically with AWS APIs.

Topics

- [Configuring AWS Secrets Manager access token authentication](#)
- [Integrating with MediaPackage endpoints that use CDN authorization](#)

- [How MediaTailor Secrets Manager access token authentication works](#)

Configuring AWS Secrets Manager access token authentication

When you want to use AWS Secrets Manager access token authentication, you perform the following steps:

1. You [create an AWS Key Management Service customer managed key](#).
2. You [create a AWS Secrets Manager secret](#). The secret contains your access token, which is stored in Secrets Manager as an encrypted secret value. MediaTailor uses the AWS KMS customer managed key to decrypt the secret value.
3. You configure an AWS Elemental MediaTailor source location to use Secrets Manager access token authentication.

The following section provides step-by-step guidance on how to configure AWS Secrets Manager access token authentication.

Topics

- [Step 1: Create an AWS KMS symmetric customer managed key](#)
- [Step 2: Create an AWS Secrets Manager secret](#)
- [Step 3: Configure a MediaTailor source location with access token authentication](#)

Step 1: Create an AWS KMS symmetric customer managed key

You use AWS Secrets Manager to store your access token in the form of a `SecretString` stored in a secret. The `SecretString` is encrypted through the use of an *AWS KMS symmetric customer managed key* that you create, own, and manage. MediaTailor uses the symmetric customer managed key to facilitate access to the secret with a grant, and to encrypt and decrypt the secret value.

Customer managed keys let you perform tasks such as the following:

- Establishing and maintaining key policies
- Establishing and maintaining IAM policies and grants
- Enabling and disabling key policies
- Rotating cryptographic key material

- Adding tags

For information about how Secrets Manager uses AWS KMS to protect secrets, see the topic [How AWS Secrets Manager uses AWS KMS](#) in the *AWS Key Management Service Developer Guide*.

For more information about customer managed keys, see [Customer managed keys](#) in the *AWS Key Management Service Developer Guide*.

 Note

AWS KMS charges apply for using a customer managed key For more information about pricing, see the [AWS Key Management Service Pricing](#) page.

You can create an AWS KMS symmetric customer managed key using the AWS Management Console or programmatically with the AWS KMS APIs.

To create a symmetric customer managed key

Follow the steps for [Creating a symmetric customer managed key](#) in the *AWS Key Management Service Developer Guide*.

Make a note of the key Amazon Resource Name (ARN); you'll need it in [Step 2: Create an AWS Secrets Manager secret](#).

Encryption context

An *encryption context* is an optional set of key-value pairs that contain additional contextual information about the data.

Secrets Manager includes an [encryption context](#) when encrypting and decrypting the `SecretString`. The encryption context includes the secret ARN, which limits the encryption to that specific secret. As an added measure of security, MediaTailor creates an AWS KMS grant on your behalf. MediaTailor applies a [GrantConstraints](#) operation that only allows us to *decrypt* the `SecretString` associated with the secret ARN contained in the Secrets Manager encryption context.

For information about how Secrets Manager uses encryption context, see the [Encryption context](#) topic in the *AWS Key Management Service Developer Guide*.

Setting the key policy

Key policies control access to your customer managed key. Every customer managed key must have exactly one key policy, which contains statements that determine who can use the key and how they can use it. When you create your customer managed key you can use the default key policy. For more information, see [Authentication and access control for AWS KMS](#) in the *AWS Key Management Service Developer Guide*.

To use your customer managed key with your MediaTailor source location resources, you must give permission to the IAM principal that calls [CreateSourceLocation](#) or [UpdateSourceLocation](#) to use the following API operations:

- `kms:CreateGrant` – Adds a grant to a customer managed key. MediaTailor creates a grant on your customer managed key that lets it use the key to create or update a source location configured with access token authentication. For more information about using [Grants in AWS KMS](#), see the *AWS Key Management Service Developer Guide*.

This allows MediaTailor to do the following:

- Call `Decrypt` so that it can successfully retrieve your Secrets Manager secret when calling [GetSecretValue](#).
- Call `RetireGrant` to retire the grant when the source location is deleted, or when access to the secret has been revoked.

The following is an example policy statement that you can add for MediaTailor:

```
{
  "Sid": "Enable MediaTailor Channel Assembly access token usage for the
MediaTailorManagement IAM role",
  "Effect": "Allow",
  "Principal": {
    "AWS": "arn:aws:iam::account number:role/MediaTailorManagement"
  },
  "Action": "kms:CreateGrant",
  "Resource": "*",
  "Condition": {
    "StringEquals": {
      "kms:ViaService": "mediatailor.region.amazonaws.com"
    }
  }
}
```



```
}
```

For more information about specifying permissions in a policy and troubleshooting key access, see [Grants in AWS KMS](#) in the *AWS Key Management Service Developer Guide*.

Step 2: Create an AWS Secrets Manager secret

Use Secrets Manager to store your access token in the form of a `SecretString` that's encrypted by an AWS KMS customer managed key. MediaTailor uses the key to decrypt the `SecretString`. For information about how Secrets Manager uses AWS KMS to protect secrets, see the topic [How AWS Secrets Manager uses AWS KMS](#) in the *AWS Key Management Service Developer Guide*.

If you use AWS Elemental MediaPackage as your source location origin, and would like to use MediaTailor Secrets Manager access token authentication follow the procedure [the section called "Integrating with MediaPackage endpoints that use CDN authorization"](#).

You can create a Secrets Manager secret using the AWS Management Console or programmatically with the Secrets Manager APIs.

To create a secret

Follow the steps for [Create and manage secrets with AWS Secrets Manager](#) in the *AWS Secrets Manager User Guide*.

Keep in mind the following considerations when creating your secret:

- The [KmsKeyId](#) must be the [key ARN](#) of the customer managed key you created in Step 1.
- You must supply a [SecretString](#). The `SecretString` should be a valid JSON object that includes a key and value containing the access token. For example, `{"MyAccessTokenIdentifier":"112233445566"}`. The value must be between 8-128 characters long.

When you configure your source location with access token authentication, you specify the `SecretString` key. MediaTailor uses the key to look up and retrieve the access token stored in the `SecretString`.

Make a note of the secret ARN and the `SecretString` key. You'll use them when you configure your source location to use access token authentication.

Attaching a resource-based secret policy

To let MediaTailor access the secret value, you must attach a resource-based policy to the secret. For more information, see [Attach a permissions policy to an Secrets Manager Secret](#) in the *AWS Secrets Manager User Guide*.

The following is a policy statement example that you can add for MediaTailor:

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "mediatailor.amazonaws.com"
      },
      "Action": "secretsmanager:GetSecretValue",
      "Resource": "arn:aws:secretsmanager:us-  
east-1:111122223333:secret:secret-name"
    }
  ]
}
```

Step 3: Configure a MediaTailor source location with access token authentication

You can configure Secrets Manager access token authentication using the AWS Management Console or programmatically with the MediaTailor APIs.

To configure a source location with Secrets Manager access token authentication

Follow the steps for [Access configuration](#) in the *AWS Elemental MediaTailor User Guide*.

Integrating with MediaPackage endpoints that use CDN authorization

If you use AWS Elemental MediaPackage as your source location origin, MediaTailor can integrate with MediaPackage endpoints that use CDN authorization.

To integrate with a MediaPackage endpoint that uses CDN authorization, use the following procedure.

To integrate with MediaPackage

1. Complete the steps in [Setting up CDN authorization](#) in the *AWS Elemental MediaPackage User Guide*, if you haven't already.
2. Complete the procedure in [the section called "Step 1: Create an AWS KMS symmetric customer managed key"](#).
3. Modify the secret that you created when you set up MediaPackage CDN authorization. Modify the secret with the following values:
 - Update the KmsKeyId with the customer managed key ARN that you created in [the section called "Step 1: Create an AWS KMS symmetric customer managed key"](#).
 - (Optional) For the SecretString, you can either rotate the UUID to a new value, or you can use the existing encrypted secret as long as it's a key and value pair in a standard JSON format, such as {"MediaPackageCDNIdentifier": "112233445566778899"}.
4. Complete the steps in [the section called "Attaching a resource-based secret policy"](#).
5. Complete the steps in [the section called "Step 3: Configure a MediaTailor source location with access token authentication"](#).

How MediaTailor Secrets Manager access token authentication works

After you create or update a source location to use access token authentication, MediaTailor includes the access token in an HTTP header when requesting source content manifests from your origin.

Here's an overview of how MediaTailor uses Secrets Manager access token authentication for source location origin authentication:

1. When you create or update a MediaTailor source location that uses access token authentication, MediaTailor sends a [DescribeSecret](#) request to Secrets Manager to determine the AWS KMS key associated with the secret. You include the secret ARN in your source location access configuration.
2. MediaTailor creates a [grant](#) for the customer managed key, so that MediaTailor can use the key to access and decrypt the access token stored in the SecretString. The grant name will be `MediaTailor-SourceLocation-your AWS account ID-source location name`.

You can revoke access to the grant, or remove MediaTailor's access to the customer managed key at any time. For more information, see [RevokeGrant](#) in the *AWS Key Management Service API Reference*.

3. When a VOD source is created or updated, or used in a program, MediaTailor makes HTTP requests to the source locations to retrieve the source content manifests associated with the VOD sources in the source location. If the VOD source is associated with a source location that has an access token configured, the requests include the access token as an HTTP header value.

Working with VOD sources

A VOD source represents a single piece of content, such as a video or an episode of a podcast, that you add to your source location. You add one or more VOD sources to your source location, and then associate each VOD source with a program after you create your channel.

Each VOD source must have at least one *package configuration*. A package configuration specifies a package format, manifest location, and source group for your VOD source. When you create your channel, you use the package configuration's source groups to create the corresponding outputs on your channel. For example, if your source is packaged in two different formats—HLS and DASH—then you'd create two package configurations, one for DASH and one for HLS. Then, you would create two channel outputs, one for each package configuration. Each channel output provides an endpoint that's used for playback requests. So, using the preceding example, the channel would provide an endpoint for HLS playback requests and an endpoint for DASH playback requests.

If you would like the offsets of ad markers in your manifest to be detected automatically, each ad marker must appear at the same offset across all package configurations and have a duration of zero. For HLS, MediaTailor will detect DATERANGE and EXT-X-CUE-OUT tags. For DASH, HLS will detect the first Event tag within each EventStream tag.

In the following example, an ad break opportunity will be detected at an offset of 12000ms because of the DATERANGE tag with a duration of 0.0. The first DATERANGE tag at an offset of 0ms will not be detected because it has a duration of 10.0.

```
#EXTM3U
#EXT-X-VERSION:3
#EXT-X-TARGETDURATION:6
#EXT-X-MEDIA-SEQUENCE:0
#EXT-X-DATERANGE:ID="1001",START-DATE="2021-09-16T23:51:05.249Z",DURATION=10.0,SCTE35-
OUT=0xFC302500000003289800FFF01405000003E97FEFFE1D381BD8FE000DBBA00001010100000FD2B275
```

```
#EXTINF:6.000,
../.../719f911124e0495cbb067c91c1d6c298/1785a16ca14d4c2884781f25333f6766/index_1_0.ts
#EXTINF:6.000,
../.../719f911124e0495cbb067c91c1d6c298/1785a16ca14d4c2884781f25333f6766/index_1_1.ts
#EXT-X-DATERANGE:ID="1001",START-DATE="2021-09-16T23:51:05.249Z",DURATION=0.0,SCTE35-
OUT=0xFC302500000003289800FFF01405000003E97FEFFE1D381BD8FE000DBBA00001010100000FD2B275
#EXTINF:6.000,
../.../719f911124e0495cbb067c91c1d6c298/1785a16ca14d4c2884781f25333f6766/index_1_2.ts
```

In the following example, an ad break opportunity will be detected at an offset of 0ms because the EXT-X-CUE-OUT tag has a duration of 0 and is followed immediately by an EXT-X-CUE-IN tag. The second EXT-X-CUE-OUT/EXT-X-CUE-IN pair will not be detected because it has a duration of 10.

```
#EXTM3U
#EXT-X-VERSION:3
#EXT-X-TARGETDURATION:6
#EXT-X-MEDIA-SEQUENCE:0
#EXT-X-CUE-OUT:0
#EXT-X-CUE-IN
#EXTINF:6.000,
../.../719f911124e0495cbb067c91c1d6c298/1785a16ca14d4c2884781f25333f6766/index_1_0.ts
#EXTINF:6.000,
../.../719f911124e0495cbb067c91c1d6c298/1785a16ca14d4c2884781f25333f6766/index_1_1.ts
#EXT-X-CUE-OUT:10
...
#EXT-X-CUE-IN
#EXTINF:6.000,
../.../719f911124e0495cbb067c91c1d6c298/1785a16ca14d4c2884781f25333f6766/index_1_2.ts
```

In the following example, an ad break opportunity will be detected at an offset of 0ms because the first Event in the EventStream occurs in the period starting at PT0.000S. The second Event in the EventStream will not be detected.

```
<Period start="PT0.000S" id="9912561" duration="PT29.433S">
<EventStream timescale="90000" schemeIdUri="urn:scte:scte35:2013:xml">
<Event duration="0">
  <scte35:SpliceInfoSection protocolVersion="0" ptsAdjustment="1241950593" tier="4095">
    <scte35:SpliceInsert spliceEventId="99" spliceEventCancelIndicator="false"
outOfNetworkIndicator="true" spliceImmediateFlag="false" uniqueProgramId="1"
availNum="1" availsExpected="1">
      <scte35:Program><scte35:SpliceTime ptsTime="3552273000"/></scte35:Program>
```

```

        <scte35:BreakDuration autoReturn="true" duration="2700000"/>
    </scte35:SpliceInsert>
</scte35:SpliceInfoSection>
</Event>
<Event duration="0">
    <scte35:SpliceInfoSection protocolVersion="0" ptsAdjustment="1241950593" tier="4095">
        <scte35:SpliceInsert spliceEventId="99" spliceEventCancelIndicator="false"
outOfNetworkIndicator="true" spliceImmediateFlag="false" uniqueProgramId="1"
availNum="1" availsExpected="1">
            <scte35:Program><scte35:SpliceTime ptsTime="3552273000"/></scte35:Program>
            <scte35:BreakDuration autoReturn="true" duration="2700000"/>
        </scte35:SpliceInsert>
    </scte35:SpliceInfoSection>
</Event>
</EventStream>
...
</Period>

```

Adding VOD sources to your source location

The following procedure explains how to add VOD sources to your source location and set up package configurations using the MediaTailor console. For information about how to add VOD sources using the MediaTailor API, see [CreateVodSource](#) in the *AWS Elemental MediaTailor API Reference*.

Important

Before you add your VOD sources, make sure that they meet these requirements:

- Source variants must all have the same length, as determined by the source manifest.
- Within a package configuration, each source must have the same number of child streams.

Because of these requirements, we don't support per title or automated ABR, because these encoding methods can produce varying manifest lengths and child streams.

We recommend that you use an encoding template that includes a minimum segment length to ensure that your encoded sources meet these requirements.

To add VOD sources to your source locations

1. Open the MediaTailor console at <https://console.aws.amazon.com/mediatailor/>.
2. In the navigation pane, choose **Channel assembly** > **Source locations**.
3. In the **Source locations** pane, choose the source location that you created in the [To create a source location](#) procedure.
4. Choose **Add VOD source**.
5. Under **VOD source details**, enter a name for your VOD source:
 - **Name:** An identifier for your VOD source, such as **my-example-video**.
6. Under **Package configurations** > *source-group-name* enter information about the package configuration:

Note

Your source's package configurations must all have the same duration, as determined by the source's manifest. And, all of the sources within a package configuration must have the same number of child streams. To meet these requirements, we recommend that you use an encoding template for your assets. We recommend that you use an encoding template with a minimum segment length of one second. MediaTailor doesn't support per title or automated adaptive bitrate streaming (ABR) because these encoding methods violate these requirements.

- **Source group:** Enter a source group name that describes this package configuration, such as HLS-4k. Make a note of this name; you'll reference it when you create your channel's output. For more information, see [Using source groups with your channel's outputs](#).
- **Type:** Select the packaged format for this configuration. MediaTailor supports HLS and DASH.
- **Relative path:** The relative path from the source location's **Base HTTP URL** to the manifest. For example, **/my/path/index.m3u8**.

Note

MediaTailor automatically imports all of the closed captions and child streams contained within a parent multivariant playlist. You don't need to create separate

package configurations for each of your sources renditions (DASH) or variant streams (HLS).

For more information about package configurations, see [Using package configurations](#).

7. Choose **Add VOD source**.

If you want to add more VOD sources, repeat steps 4-7 in the procedure.

Working with live sources

A *live source* represents a single live stream, such as a live football game or news broadcast, that you add to your source location. After you create your channel, you add one or more live sources to your source location, and then associate each live source with a program.

MediaTailor supports these types of linear channel assembly:

- VOD sources for a channel that contains VOD-to-live content
- Live sources for a channel that contains live-to-live content intermixed with VOD-to-live content

An example of VOD-to-live content is a channel that assembles a library of VOD assets into a live stream. One example of live-to-live content mixed with VOD-to-live content is a channel that shows mostly VOD content, except for a nightly news event or a pre-scheduled live sporting event. Another example of live-to-live content mixed with VOD-to-live content is an all live-to-live channel with origins that vary based on the time of day.

You can use live sources to set up a regional channel that shows mostly national programming, but also includes regional programming overrides, and has VOD content mixed in. To do so, you run one encoder/packager pair for the national content, then run regional encoders when those regions are live. Then, you create regional channel-assembly channels, each with their own schedules. This way, viewers can switch back and forth as needed. This setup helps you minimize encoding/packaging costs.

Each live source must have at least one package configuration. A *package configuration* specifies a package format, manifest location, and source group for your live source. When you create your channel, you use the package configuration's source groups to create the corresponding outputs on your channel. For example, if your source is packaged in two different formats—HLS and DASH—then you'd create two package configurations, one for DASH and one for HLS. Then, you'd

create two channel outputs, one for each package configuration. Each channel output provides an endpoint that's used for playback requests. In this example, the channel provides an endpoint for HLS playback requests and an endpoint for DASH playback requests.

General requirements for using live sources

When you use live sources, your content must align with the following general requirements:

- HLS live sources - You must provide #EXT-X-PROGRAM-DATE-TIME tags for the first segment in the manifest window, and on every discontinuity.
- HLS - You must configure ad markers as DATERANGE.
- Source manifest window - We recommend using a manifest window with a duration that's at least as long as the manifest window on your MediaTailor Channel Assembly channel. As a best practice, consider using a manifest window duration that's 30 seconds or longer than the manifest window on the Channel Assembly channel.
- Make the target duration match the duration of the existing sources.
- Make the number of child playlists match that of the existing sources.

Configurations

If you use other AWS Elemental media services as part of your live sources workflow, we recommend following best practices when setting up your MediaPackage configuration. The following table describes how to configure MediaPackage settings based on the streaming standard you use.

MediaPackage setup for live sources

Standard	Setting	Value	Necessity	Notes
HLS	Endpoint type	Apple HLS	Required unless using CMAF	To match HLS ts AWS Elemental MediaConvert jobs
HLS	Endpoint type	CMAF	Required unless using Apple HLS	To match HLS mp4 AWS Elemental

Standard	Setting	Value	Necessity	Notes
				MediaConvert jobs
HLS	ProgramDateTimeIntervalSeconds	1	Required	You must specify #EXT-X-PROGRAM-DATE-TIME on every segment in order to prevent playback issues when there are discontinuities.
HLS	PlaylistWindowSeconds	30 seconds longer than the channel assembly manifest window	Required	
HLS	AdMarkers	DATERANGE	Required when passing through ad markers	
HLS	IncludeIframeOnlyStream	Disabled	Recommended	
DASH	ManifestLayout	FULL	Recommended	

Standard	Setting	Value	Necessity	Notes
DASH	SegmentTemplateFormat	NUMBER_WITH_TIMELINE or TIME_WITH_TIMELINE	Recommended	NUMBER_WITH_DURATION is not supported.
DASH	ManifestWindowSeconds	30 seconds longer than the channel assembly manifest window	Required	
DASH	PeriodTriggers	ADS	Required when passing through ad markers	

Adding live sources to your source location

The following procedure explains how to use the MediaTailor console to add live sources to your source location and set up package configurations. For information about how to add live sources using the MediaTailor API, see [CreateLiveSource](#) in the *AWS Elemental MediaTailor API Reference*.

Important

Before you add your live sources, make sure that within a package configuration, each source has the same number of child streams.

To add live sources to your source locations

1. Open the MediaTailor console at <https://console.aws.amazon.com/mediatailor/>.
2. In the navigation pane, choose **Channel assembly** > **Source locations**.
3. In the **Source locations** pane, choose the source location that you created in the [To create a source location](#) procedure.

4. On the **Live sources** tab, choose **Add live source**.
5. Under **live source details**, enter a name for your live source:
 - **Name:** An identifier for your live source, such as **my-example-video**.
6. Under **Package configurations** > *source-group-name* enter information about the package configuration:

 Note

Within a package configuration, all of the VOD sources and live sources must have the same number of child streams. We recommend that you configure your source streams the same way.

- **Source group:** Enter a source group name that describes this package configuration, such as HLS-4k. Make a note of this name; you'll reference it when you create your channel's output. For more information, see [Using source groups with your channel's outputs](#).
- **Type:** Select the packaged format for this configuration. MediaTailor supports HLS and DASH.
- **Relative path:** The relative path from the source location's **Base HTTP URL** to the manifest. For example, **/my/path/index.m3u8**.

 Note

MediaTailor automatically imports all of the closed captions and child streams contained within a parent multivariant playlist. You don't need to create separate package configurations for each of your sources renditions (DASH) or variant streams (HLS).

For more information about package configurations, see [Using package configurations](#).

7. Choose **Add live source**.

If you want to add more live sources, repeat steps 4-6 in the procedure.

Using package configurations

A package configuration is a representation of the source that contains the various packaging characteristics required for playback on different devices. For example, you might have a source that has three packaged formats: HLS with DRM, DASH with segment timeline addressing, and HLS with CMAF segments.

Channel assembly doesn't repackage your sources. If you want to include multiple packaged formats for a given source, you must make each packaged format available at the source location and specify the path to each packaged format.

Each package configuration object must include the following:

- **Relative path** - The full path to the source's packaged format, relative to the source location. For example, `/my/path/index.m3u8`.
- **Source group** - The name of the source group used to associate package configurations with a channel's output.
- **Type** - Either HLS or DASH.

After you have created a channel, you must also declare each source group that you want to use for the channel's output.

Manifest caching

MediaTailor periodically and opportunistically caches source playlists to improve channel assembly performance and reliability. Sometimes, the cached version becomes stale compared to the origin version at your source location. To force MediaTailor to refresh the cached version of the source, call [UpdateVodSource](#). For example, use this call when the embedded paths change in your source. Make sure that you always keep an up-to-date version of the source available on your source location, even if you see few requests from MediaTailor.

Working with channels

A channel assembles your source manifests into a linear stream. Each channel contains one or more outputs that correspond to your package configurations.

First you create a channel, then you add your VOD sources and live sources to the channel's schedule by creating *programs*. Each program is associated with a VOD source or a live source.

Topics

- [Create a channel using the MediaTailor console](#)
- [Using source groups with your channel's outputs](#)
- [Delete a channel using the MediaTailor console](#)

Create a channel using the MediaTailor console

The following procedure describes how to create a channel using the MediaTailor console.

To create a channel

1. Open the MediaTailor console at <https://console.aws.amazon.com/mediatailor/>.
2. In the navigation pane, choose **Channel assembly** > **Channels**.
3. On the navigation bar, choose **Create channel**.
4. Under **Channel details**, enter details about your channel:
 - **Name:** Enter a name for your channel.
 - **Tier:** The tier determines what features the channel supports and how much it costs to run the channel. For more information about pricing, see the [Channel Assembly pricing page](#). MediaTailor supports the following tiers:
 - **Basic** - The Basic tier supports both the Linear and Loop playback modes, and does not support live sources.
 - **Standard** - The Standard tier supports live sources, and requires the Linear playback mode.

When you select **Standard** in **Channel details**, you can define the audiences under **Audiences details**. These audiences will be used for programRules when you are going to create audienceMedia for your default program..

- Choose **Add**.
- Enter the **Audience** name in the text box. It must be between 1 and 32 alphanumeric characters long.
- Choose **Confirm**.
- Choose **Next**.
- **Playback mode:** The playback mode sets the channel's playback behavior. MediaTailor supports the following playback modes:

- **Loop** - The programs in the schedule play back-to-back in an endless loop. After the last program plays in a schedule, playback loops back to the first program. Playback continues looping until you stop the channel.
- **Linear** - Each program in the schedule plays once, back-to-back.

5.

For **Filler slate**, select the **Source location name** referencing the slate location, and the **VOD source name** to use as slate. MediaTailor uses the slate to fill gaps between programs in the schedule. If the duration of the slate is less than the duration of the gap between programs, MediaTailor loops the slate. You must configure filler slate field is if your channel uses the linear playback mode. MediaTailor doesn't support filler slate for the loop playback mode.

6. Choose **Next**.

7. Specify audience details under program rules.

8. When you select **Standard** in **Channel details**, you can define the audiences under **Audiences** details. These audiences will be used for **programRules** when you are going to create **audienceMedia** for your default program:

- Select **Add**, and then add an Audience in the text box, then select **Confirm**.

 Note

Enter a name that's no longer than_ 32 alphanumeric characters.

- **Output type**: Select the streaming format for the channel. DASH and HLS are supported.
- **Source group**: Enter the name of the source group that you created in your package configuration, as described in [Adding VOD sources to your source location](#).

9. Select **Next**.

10. Under **Manifest settings**, enter additional information about your manifest settings:

- **Manifest window (sec)**: The time window, in seconds, contained in each manifest. The minimum value is 30 seconds, and the maximum value is 3600 seconds.
- **Ad markup type(HLS outputs only)**: The type of ad tags that appear in VOD program ad breaks. Select **Daterange** to have MediaTailor insert ad breaks into VOD programs with EXT-X-DATERANGE tags. Select **Scte35 Enhanced** to have MediaTailor insert ad breaks into VOD programs using EXT-X-CUE-OUT and EXT-X-CUE-IN tags. For more information about these tag types, see [SCTE-35 messages for ad breaks](#). For live workflows, MediaTailor

always passes through DATERANGE tags, and doesn't pass through any Enhanced SCTE35 tags, regardless of the selected Ad markup type.

11. If you want to configure multiple channel outputs, under **Outputs** choose **Add**. Then, configure the details for your output by completing the steps 6 and 7 in this procedure.
12. Choose **Next**.
13. Under **Channel policy**, choose your channel's IAM policy settings:
 - **Do not attach channel policy**: Restrict playback to only those who have access to this account's credentials.
 - **Attach custom policy**: Define your own policy and restrict access to as few or as many as you want.
 - **Attach public policy**: Accept all incoming client requests to a channel's output. You must use this option if you want to use MediaTailor ad insertion.
14. Choose **Next**.
15. Review your settings on the **Review and create** pane.
16. Choose **Create channel**.

 Note

Channels are created in a stopped state. Your channel won't be active until you start it with the MediaTailor console or the MediaTailor StartChannel API.

Using source groups with your channel's outputs

A source group associates a package configuration with an output on a channel. When you create the package configuration on the source, you identify the source group's name. Then, when you create the output on the channel, you enter that same name to associate the output with the package configuration. VOD sources and live sources that are added to a program on a channel must belong to the source group that's identified in the output.

For example:

- VOD sources 1 and 2 both have three package configurations that have the source groups: **HLS**, **DASH**, and **HLS-4k**.
- VOD source 3 has two package configurations with source groups **HLS** and **DASH**.

If channel A has two outputs with source groups **HLS** and **DASH**, the channel output can use all three VOD sources. That's because VOD sources 1, 2, and 3 all have package configurations with source group labels **HLS** and **DASH**.

If channel B has two outputs with source groups **HLS** and **HLS-4k**, it can use VOD source 1 and 2, but not 3. This is because VOD sources 1 and 2 both have package configurations with source group labels **HLS** and **HLS-4k**.

If channel C has a single output with the source group **DASH**, it can use all three VOD sources. All three VOD sources have package configurations with the **DASH** source group.

Delete a channel using the MediaTailor console

To delete your channel, complete the following procedure.

To delete your channel

1. Open the MediaTailor console at <https://console.aws.amazon.com/mediatailor/>.
2. In the navigation pane, choose **Channel assembly** > **Channels**.
3. Select the channel that you want to delete.
4. If your channel is running, from the **Actions** drop-down menu, choose **Stop**. You must stop your channel before you can delete it.
5. When your channel is stopped, from the **Actions** drop-down menu, choose **Delete**.

Adding a program to a channel's schedule

Each program contains a VOD source or a live source that's part of a source location in your account. You add your programs to your channel's schedule to control the order that they play in your channel's stream.

A program that contains a VOD source can be configured with one or more ad breaks. Each ad break contains a slate, which is a VOD source from a source location. To create the ad break, you add the slate at an offset in milliseconds into the program.

Topics

- [Creating a program within a channel schedule using the MediaTailor console](#)

- [Defining audience cohorts and alternate content with Program Rules](#)
- [Generating audience-specific manifests](#)

Creating a program within a channel schedule using the MediaTailor console

The following procedure describes how to create a program within your channel's schedule using the MediaTailor console. It also describes how to configure ad breaks, which are optional. For information about how to create programs using the MediaTailor API, see [CreateProgram](#) in the *AWS Elemental MediaTailor API Reference*.

To add a program

1. Open the MediaTailor console at <https://console.aws.amazon.com/mediatailor/>.
2. In the navigation pane, choose **Channel assembly** > **Channels**.
3. In the **Channels** pane, choose the channel that you created in the [To create a channel](#) procedure.
4. In the **Program details** enter details about your program:
 - **Name:** This is the name of the program that you add to your channel.
 - **Source type:** Determines what type of source video the program plays. This option is only available for Standard channels.
 - **VOD** - The program plays a video-on-demand source, such as a pre-recorded TV episode.
 - **Live** - The program plays a live source, such as a live news broadcast.
 - **Source location name:** The source location that MediaTailor associates with the program.
 - If you choose **Select an existing source location**, choose a sourcelocation name from the **Select a source location** menu. Alternatively, search for your source location by name. This is helpful if you have a large number of source locations.
 - If you choose **Enter the source location name**, search for your source location by name.
 - **VOD source name:** The name of the VOD source that MediaTailor associates with the program:
 - If you choose **Select an existing VOD source**, select a VOD source name from the list of VOD sources that are associated with your account. Alternatively, search for your VOD source by name. This is helpful if you have a large number of VOD sources.

- If you choose **Search by name**, search for your live source by name.
 - **Live source name:** The name of the live source to be associated with the program. This option is only available if you selected **Live** as the source type.
 - If you choose **Select an existing source location**, choose a source location name from the **Select a source location** menu. Alternatively, search for your source location by name. This is helpful if you have a large number of source locations.
 - If you choose **Enter the source location name**, search for your source location by name.
 - **VOD source name:** The name of the VOD source that MediaTailor associates with the program:
 - If you choose **Select an existing live source**, select a live source name from the list of live sources that are associated with your account. You can alternatively search for your live source by name. This is helpful if you have a large number of live sources.
 - If you choose **Search by name**, search for your live source by name.
5. Select **Next** to go to the **Schedule Configuration** tab.
6. Under **Playback configuration**, define when a program plays in your channel's schedule:
- **Duration in milliseconds:** Defines the duration of the program in milliseconds. This option is only available for programs that use live sources.
 - **Transition type:** Defines the transitions from program to program in the schedule:
 - **Relative:** The program plays either before or after another program in the schedule. This option is only available for programs that use VOD sources.
 - **Absolute:** The program plays at a specific wall-clock time. MediaTailor makes a best effort to play the program at the clock time that you specify. MediaTailor starts playback of the program on a common segment boundary between the preceding program or slate. This option is only available for channels configured to use the linear [Playback mode: The playback mode sets the channel's playback behavior. MediaTailor supports the following playback modes:](#).
 - **Program start time:** For absolute transition types, the wall-clock time when the program is scheduled to play. If you are adding this program to a running linear channel, you must enter a start time that's 15 minutes or later from the current time.
 - **Relative position:** Choose where to insert the program into the schedule, relative to another program. You can select **Before program** or **After program**. This setting does not apply if this is the first program in your channel's schedule.

- If you choose **Select an existing program**, select the program name from a predefined list of the next 100 programs played by the channel from the **Use existing program** menu.
- If you choose **Search for a program by name**, enter the name of an existing program in your channel.

If you'd like to add ad breaks to your program, continue to the next step. Ad breaks are only configurable for programs that use VOD sources. For live sources, ad breaks in DASH manifests and ad breaks in HLS manifests that use the EXT-X-DATERANGE tag are passed through automatically.

7. Select **Next** to go to **Add ad breaks**.
8. Select **Add ad break**. Under **Ad breaks**, configure the settings for the ad break:
 - **Slate source location name:** Choose **Select an existing source location** and choose the source location where your slate is stored that you created earlier in this task.
 - **VOD source name:** Choose **Select an existing VOD source** and choose the VOD source you're using for slate that you added earlier in this task. The duration of the slate determines the duration of the ad break.
 - **Offset in milliseconds:** This value determines the ad break start time in milliseconds, as an offset relative to the beginning of the program. Enter any value that's less than the duration of the VOD source, and that aligns with a segment boundary on all tracks within the program's VOD source (all audio, video and closed caption tracks), otherwise the ad break will be skipped. For example, if you enter **0**, this creates a pre-roll ad break that plays before the program begins.
 - **Avail number:** MediaTailor writes this value is written to `splice_insert.avail_num`, as defined in section 9.7.3.1. of the SCTE-35 specification, [Digital Program Insertion Cueing Message](#). The default value is 0. Values have to be between 0 and 256, inclusive.
 - **Avail expected:** MediaTailor writes this value to `splice_insert.avails_expected`, as defined in section 9.7.3.1. of the SCTE-35 specification. The default value is 0. Values have to be between 0 and 256, inclusive.
 - **Splice event ID:** MediaTailor writes this value to `splice_insert.splice_event_id`, as defined in section 9.7.3.1. of the SCTE-35 specification. The default value is 1.

- **Unique program ID:** MediaTailor writes this value to `splice_insert.unique_program_id`, as defined in section 9.7.3.1. of the SCTE-35 specification. The default value is 0. Values have to be between 0 and 256, inclusive.

9. For a Standard Linear Channel, select **Next** to go to **Set alternate media**.

For more information on using MediaTailor to create alternate media, see [Creating alternate media](#).

For more advanced information on using MediaTailor to personalize your ad breaks, see [Insert personalized ads and ad breaks in a channel stream](#).

10. Select **Next** to go to **Review and create**.

11. Select **Add program**.

For more advanced information on using MediaTailor to personalize your ad breaks, see [Insert personalized ads and ad breaks in a channel stream](#).

12.

 Important

For looping channels, if you modify the program list for a program that is scheduled within the next 10 minutes, the edit won't become apparent until the next loop.

Under **Program details**, enter details about your program:

- **Name:** This is the name of the program that you add to your channel.
- **Source type:** Determines what type of source the program plays. This option is only available for Standard channels.
 - **VOD** - The program plays a VOD source, such as a pre-recorded TV episode.
 - **Live** - The program plays a live source, such as a live news broadcast.
- **Source location name:** The source location to be associated with the program.

If you choose **Select an existing source location**, select a source location name from the **Select a source location** drop-down menu. You can alternatively search for your source location by name. This is helpful if you have a large number of source locations.

If you choose **Enter the source location name**, search for your source location by name.

- **VOD source name:** The name of the VOD source to be associated with the program.

If you choose **Select an existing VOD source**, select a VOD source name from the list of VOD sources that are associated with your account. You can alternatively search for your VOD source by name. This is helpful if you have a large number of VOD sources.

If you choose **Search by name**, search for your VOD source by name.

- **Live source name:** The name of the live source to be associated with the program. This option is only available if you selected **Live** as the source type.

If you choose **Select an existing live source**, select a live source name from the list of live sources that are associated with your account. You can alternatively search for your live source by name. This is helpful if you have a large number of live sources.

If you choose **Search by name**, search for your live source by name.

13. Under **Playback configuration**, define when a program plays in your channel's schedule:

- **Duration in milliseconds:** Defines the duration of the program in milliseconds. This option is only available for programs that use live sources.
- **Transition type:** Defines the transitions from program to program in the schedule.
 - **Relative** - The program plays either before or after another program in the schedule. This option is only available for programs that use VOD sources.
 - **Absolute** - The program plays at a specific wall clock time. MediaTailor makes a best effort to play the program at the clock time that you specify. We start playback of the program on a common segment boundary between the preceding program or slate. This option is only available for channels configured to use the [linear playback mode](#).

 Note

Be aware of the following behavior for absolute transition types:

- If the preceding program in the schedule has a duration that extends beyond the wall clock time, MediaTailor truncates the preceding program on the common segment boundary closest to the wall clock time.
- If there are gaps between programs in the schedule, MediaTailor plays [filler slate](#). If the duration of the slate is less than the duration of the gap, MediaTailor loops the slate.

- **Program start time** - For absolute transition types, the wall clock time when the program is scheduled to play. If you are adding this program to a running linear channel, you must enter a start time that's 15 minutes or later from the current time.
- **Relative position**: Choose where to insert the program into schedule relative to another program. You can select **Before program** or **After program**. This setting does not apply if this is the first program in your channel's schedule.
- **Relative program**: The name of the program to be used to insert the new program before or after. This setting does not apply if this is the first program in your channel's schedule.

If you choose **Select an existing program**, select the program name from a predefined list of the next 100 programs played by the channel in the **Use existing program** drop-down menu.

If you choose **Search for a program by name**, enter name of an existing program in your channel.

If you'd like to add ad breaks to your program, continue to the next step. Ad breaks are only configurable for programs that use VOD sources. For live sources, ad breaks in DASH manifests and ad breaks in HLS manifests that use the EXT-X-DATERANGE tag are passed through automatically.

14. Select **Add ad break**. Under **Ad breaks**, configure the settings for the ad break:

- **Slate source location name**: Choose **Select an existing source location** and choose the source location where your slate is stored that you created earlier in this tutorial.
- **VOD source name**: Choose **Select an existing VOD source** and choose the VOD source you're using for slate that you added earlier in this tutorial. The duration of the slate determines the duration of the ad break.
- For **Offset in milliseconds**: This value determines the ad break start time in milliseconds, as an offset relative to the beginning of the program. Enter any value that's less than the duration of the VOD source, and that aligns with a segment boundary on all tracks within the program's VOD source (all audio, video and closed caption tracks), otherwise the ad break will be skipped. For example, if you enter **0**, this creates a pre-roll ad break that plays before the program begins.

Note

If MediaTailor detects ad markers, such as `asDATERANGE` or `EXT-X-CUE-OUT` for HLS and `EventStream` for DASH, with durations of zero within your VOD source, you can select the offset of those ad markers from the drop-down menu to be used as the ad break's offset. In order for an ad opportunity to be detected, it must be present at the same offset across all package configurations within a VOD source, and its duration must be zero.

- **Message type:** The SCTE-35 ad insertion type. Choose either **SPLICE_INSERT** or **TIME_SIGNAL**:
 - **SPLICE_INSERT:** Provides basic metadata about the ad break using splice insert parameters.
 - **TIME_SIGNAL:** Provides more advanced metadata using segmentation descriptors. For more information about the differences between message types, see [the section called "SCTE-35 messages for ad breaks"](#).

For **SPLICE_INSERT** message type:

- For **Avail number**, this is written to `splice_insert.avail_num`, as defined in section 9.7.3.1. of the SCTE-35 specification. The default value is 0. Values have to be between 0 and 256, inclusive.
- For **Avail expected**, this is written to `splice_insert.avails_expected`, as defined in section 9.7.3.1. of the SCTE-35 specification. The default value is 0. Values have to be between 0 and 256, inclusive.
- For **Splice event ID**, this is written to `splice_insert.splice_event_id`, as defined in section 9.7.3.1. of the SCTE-35 specification. The default value is 1.
- For **Unique program ID**, this is written to `splice_insert.unique_program_id`, as defined in section 9.7.3.1. of the SCTE-35 specification. The default value is 0. Values have to be between 0 and 256, inclusive.

For **TIME_SIGNAL** message type:

- For **Segmentation event ID**, this is written to `segmentation_descriptor.segmentation_event_id`, as defined in section 10.3.3.1 of the SCTE-35 specification. The default value is 1.

- For **Segmentation type ID**, this is written to `segmentation_descriptor.segmentation_type_id`, as defined in section 10.3.3.1 of the SCTE-35 specification. The default value is 48 (0x30). Values have to be between 0 and 256, inclusive.
- For **Segmentation UPID**, this is written to `segmentation_descriptor.segmentation_upid`, as defined in section 10.3.3.1 of the SCTE-35 specification. The value must be a hexadecimal string containing characters 0-9 and A-F. The default value is an empty string.
- For **Segmentation UPID type**, this is written to `segmentation_descriptor.segmentation_upid_type`, as defined in section 10.3.3.1 of the SCTE-35 specification. The default value is 14 (0x0E). Values have to be between 0 and 256, inclusive.
- For **Segment number**, this is written to `segmentation_descriptor.segment_num`, as defined in section 10.3.3.1 of the SCTE-35 specification. The default value is 0. Values have to be between 0 and 256, inclusive.
- For **Segments expected**, this is written to `segmentation_descriptor.segments_expected`, as defined in section 10.3.3.1 of the SCTE-35 specification. The default value is 0. Values have to be between 0 and 256, inclusive.

15. Choose **Add program**.

For more advanced information using MediaTailor to personalize your ad breaks, see [Insert personalized ads and ad breaks in a channel stream](#).

Note

If your channel has at least one output with an Enhanced Scte35 Ad markup type, you can submit ad-break metadata. MediaTailor writes the submitted key-value pairs to the EXT-X-ASSET tag for your ad break.

Defining audience cohorts and alternate content with Program Rules

With Program Rules, you can define audience cohorts for a channel and specify alternate media to play for those audiences. You can associate one or more alternate content sources with an audience

for a program. After the program ends, the default audience content will play unless you specify further alternate media.

Program Rules are available on STANDARD tier channels with the LINEAR playback mode. MediaTailor channels support alternate media for all VOD sources and live sources.

For an example use, see [Using program rules with AWS MediaTailor](#).

Defining audiences

Define audiences on a channel by typing audience one by one when configuring a MediaTailor channel. You can do this through either the MediaTailor console or the MediaTailor `CreateChannel` API. Each audience must be between 1 and 32 alphanumeric characters long. If the values provided for the audiences are invalid, then the request fails.

You can only define audiences on STANDARD tier channels with the LINEAR playback mode.

When you need to update the audiences, you can do this using either the MediaTailor console or the MediaTailor `UpdateChannel` API.

If you are using the `ProgramRules` feature, make sure that the `AudienceMedia` defined in `CreateProgram` or `UpdateProgram` request contain the existing audience defined in the channel.

Creating alternate media

The following task explains how to define alternate media using the MediaTailor console. For information about how to define alternate media using the MediaTailor API, see [CreateProgram](#) in the *AWS Elemental MediaTailor API Reference*.

To define alternate media on a new program:

1. Open the MediaTailor console at <https://console.aws.amazon.com/mediatailor/>.
2. In the navigation pane, select **Channel assembly** > **Channels**.
3. Select the channel name to which you want to add alternate media.
4. Create a program. For more information, see [Creating a program within a channel schedule using the MediaTailor console](#).
5. Configure alternate media:
 - Select **Add** in the **Audiences** box to select the audience for which you are defining alternate media.

- Select an audience defined on the channel from the **Audience** menu.
- Select **Add alternate media** to begin defining alternate media for the program.
- MediaTailor creates an **Alternate media 1** box. This is the first content that MediaTailor plays as alternate media on the program.
- Within the **Alternate media 1** box:
 - Select a **Source Location**.
 - Select either a **VOD** or **Live** Source Type:

For VOD

- Select VOD for the **Source Type**.
- (Optional) specify a **Clip Range**. With VOD Sources, including alternate-media VOD sources, you can specify a portion of a VOD source to play, clipping from the start and/or the end of the source. Specify The start and end offsets are in milliseconds.
- (Optional) a\Add Ad Breaks. This is done in the same way as when creating programs. For more information, see [Creating a program within a channel schedule using the MediaTailor console](#).

For Live

- Select Live for the **Source Type**.
- Select a **Live source**.
- Enter a **Start time** in milliseconds of the wal-clock time that this live source should start. The live source will only play within the time frame of the default program it is being defined on. If the start time is prior to the start of the default program, it will not begin until the default program does. If the start time is after the default program ends, MediaTailor will not play the live source.
- Enter a **Duration** in milliseconds. The duration must be at least 10 minutes in length.
- Additional alternate media can be added to this program for the audience by selecting **Add alternate media** again. This will create another box labeled **Alternate media 2**. You can specify up to 5 alternate-media sources per program, per audience.
- Once you are finished defining alternate media for all desired audiences, select **Next** and continue creating the program.

For more information, see [Creating a program within a channel schedule using the MediaTailor console](#).

Note

Alternate media only plays in the time frame of the program it is defined on. If all the alternate content overruns the default content, MediaTailor will truncate it. MediaTailor plays alternate media in the order in which it is defined. Live alternate-media start times will always take precedence and will truncate previously scheduled VOD sources or live sources. Any time that is not filled with alternate media for an audience will be filled with the channel-defined filler slate

- To define audience media for other audiences, select **Add** once again next to **Audiences**. Select the newly created audience, set the audience id and add alternate media as described above. Up to 5 audiences can have alternate media on any one program.

Generating audience-specific manifests

To retrieve a manifest for a particular audience, use the `aws.mediatailor.channel.audienceId` query parameter. This query parameter may be dynamically appended by your CDN or added through a call to your content or customer-management system. You must maintain the association of a given playback session to an `audienceId` outside of MediaTailor. This will retrieve an audience-specific manifest with any alternate media defined for that audience in place of default content. It is important that once a manifest is requested for a particular audience that the player always requests the manifest with that same audience ID or there could be playback errors.

If a request is made for an audience that does not exist on the channel, MediaTailor returns a 404 error.

Example Getting a manifest for an audience

```
https:// prefix>.channel-assembly.mediatailor.us-  
west-2.amazonaws.com/v1/channel/ExampleChannel/index_dash.mpd?  
aws.mediatailor.channel.audienceId=Seattle
```

Insert personalized ads and ad breaks in a channel stream

With MediaTailor, you can monetize channel assembly linear streams by inserting ad breaks in your programs without conditioning the content with SCTE-35 markers. You can use channel assembly with the MediaTailor ad insertion service, or with any server-side ad insertion (SSAI).

The following topics show how to insert personalized ads and ad breaks in your channel's linear stream.

Topics

- [Setting up ad insertion with MediaTailor](#)
- [SCTE-35 messages for ad breaks](#)

Setting up ad insertion with MediaTailor

To insert personalized ads into your channel's stream, your channel's endpoint URL is the content source for AWS Elemental MediaTailor. This guide shows how to set up MediaTailor for ad insertion.

Prerequisites

Before you begin, make sure that you meet the following requirements:

- Prepare your HLS and DASH streams for MediaTailor ad insertion.
 - If you haven't prepared content streams already, see [Step 2: Prepare a stream](#) in the *Getting started with MediaTailor ad insertion* topic.
- Have an ad decision server (ADS).
- Configure **Ad break** settings in the program. For more information, see the [Configuring ad breaks for your program](#) procedure.

As a best practice, consider using a content delivery network (CDN) in between channel assembly and MediaTailor ad insertion. The MediaTailor ad insertion service can generate additional origin requests. Therefore, it's a best practice to configure your CDN to proxy the manifests from channel assembly, then use the CDN prefixed URLs at the content source URL.

Configure MediaTailor for ad insertion

The following shows how to configure MediaTailor console settings so that you can insert personalized ads into your channel's stream.

To configure MediaTailor for ad insertion

1. Open the MediaTailor console at <https://console.aws.amazon.com/mediatailor/>.
2. In the navigation pane, choose **Configurations**.
3. Under **Required settings**, enter the basic required information about your configuration:
 - **Name:** The name of your configuration.
 - **Content source:** Enter the playback URL from your channel's output, minus the file name and extension. For advanced information about MediaTailor configuration, see [Required settings](#).
 - **Ad decision server:** Enter the URL for your ADS.
4. You can optionally configure the **Configuration aliases**, **Personalization details**, and **Advanced settings**. For information about those settings, see [Optional configuration settings](#).
5. On the navigation bar, choose **Create configuration**.

Now that you've set up MediaTailor for ad insertion, you can also set up ad breaks. For detailed instructions, see [Getting started with MediaTailor ad insertion](#).

SCTE-35 messages for ad breaks

With MediaTailor, you can create a content channel based off of source location and VOD source resources. You can then set up one or more ad breaks for each of the programs on a channel's schedule. You use messages based on the SCTE-35 specification to condition the content for ad breaks. For example, you can use SCTE-35 messages to provide metadata about the ad breaks. For more information about the SCTE-35 specification, see [Digital Program Insertion Cueing Message](#).

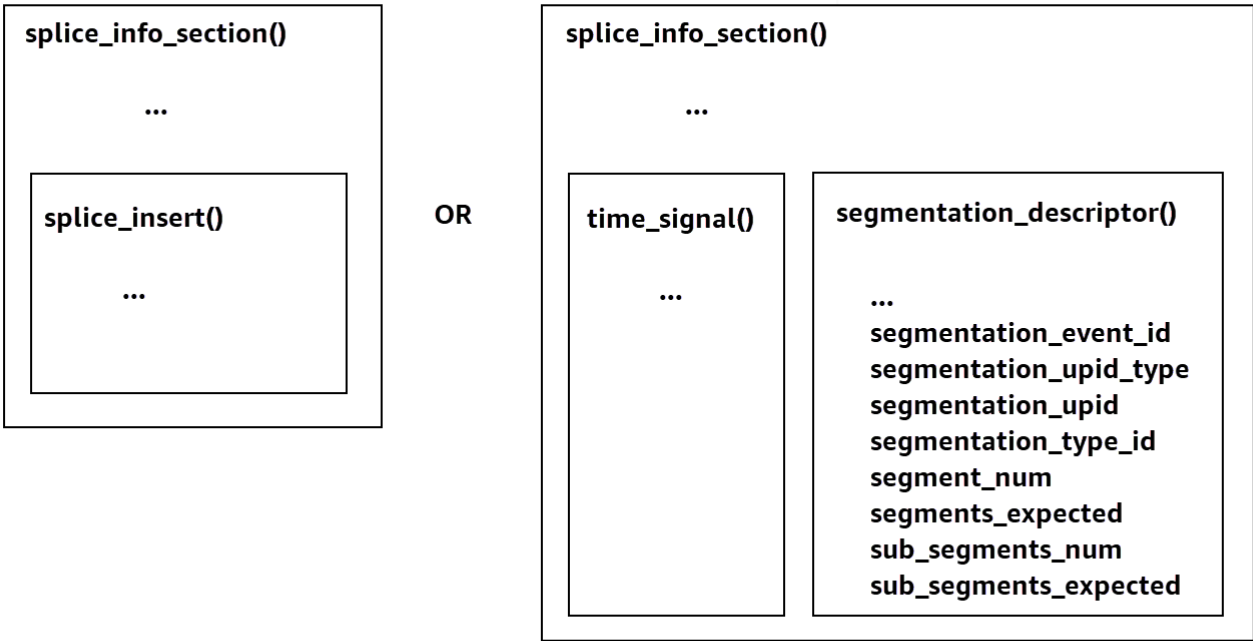
You set up the ad breaks in one of two ways:

- Attaching a `time_signal` SCTE-35 message with a `segmentation_descriptor` message. This `segmentation_descriptor` message contains more advanced metadata fields, like content identifiers, which convey more information about the ad break. MediaTailor writes the ad metadata to the output manifest as part of the EXT-X-DATERANGE (HLS) or EventStream (DASH) ad marker's SCTE-35 data.

- Attaching a `splice_insert` SCTE-35 message that provides basic metadata about the ad break.
- HLS:
 - When the Ad markup type is `Daterange`, MediaTailor specifies ad breaks as `EXT-X-DATERANGE` tags in the manifest.
 - When the Ad markup type is `Scte35 Enhanced`, MediaTailor specifies ad breaks using the following tags:
 - MediaTailor places an `EXT-X-CUE-OUT` on the first segment of the ad slate, indicating a cut from content to the ad break. It contains the expected duration of ad break, such as `EXT-X-CUE-OUT:Duration=30`.
 - `>EXT-X-ASSET`: This tag appears on the same segment as `EXT-X-CUE-OUT` and contains the ad-break metadata supplied in the `AdBreak` when the program is created or updated. It always contains CAID.
 - `EXT-0ATCLS-SCTE35`: This tag appears on the same segment as `EXT-X-CUE-OUT` and contains base64-encoded bytes of the SCTE-35 message.
 - `EXT-X-CUE-OUT-CONT`: This tag appears on each subsequent segment within the ad slate, and contains duration and elapsed-time information. It also contains the base64-encoded SCTE-35 message, and the CAID.
 - `EXT-X-CUE-IN`: This tag appears on the first segment of content after the ad break is over, and indicates a cut from an ad break back to content.

The following illustration shows the two ways of setting up ad breaks in a channel using SCTE-35 messages:

- Use a `splice_insert()` message to set up ad breaks with basic metadata.
- Use a `time_signal()` message together with a `segmentation_descriptor()` message to set up ad breaks with more detailed metadata.



For information about using `time_signal`, see section 9.7.4 of the 2022 SCTE-35 specification, [Digital Program Insertion Cueing Message](#).

The ad break information appears in the output `splice_info_section` SCTE-35 data. With MediaTailor, you can pair a single `segmentation_descriptor` message together with a single `time_signal` message.

 **Note**

If you send a `segmentation_descriptor` message, you must send it as part of the `time_signal` message type. The `time_signal` message contains only the `splice_time` field that MediaTailor constructs using a given timestamp.

The following table describes the fields that MediaTailor requires for each `segmentation_descriptor` message. For more information, see section 10.3.3.1 of the 2022 SCTE-35 specification, which you can purchased at the [ANSI Webstore website](#).

Required fields for a segmentation_descriptor message

Field	Type	Default value	Description
segmentation_event_id	integer	1	This is written to segmentation_descriptor.segmentation_event_id .
segmentation_upid_type	integer	14 (0x0E)	This is written to segmentation_descriptor.segmentation_upid_type . The value must be between 0 and 256, inclusive.
segmentation_upid	string	"" (empty string)	This is written to segmentation_descriptor.segmentation_upid . The value must be a hexadecimal string, containing characters 0-9 and A-F.
segmentation_type_id	integer	48 (0x30)	This is written to segmentation_descriptor.segmentation_type_id . The value must be

Field	Type	Default value	Description
			between 0 and 256, inclusive.
segment_num	integer	0	This is written to segmentat ion_descr iptor.segment_num . The value must be between 0 and 256, inclusive.
segments_expected	integer	0	This is written to segmentat ion_descr iptor.segments_expected . The value must be between 0 and 256, inclusive.
sub_segment_num	integer	null	This is written to segmentat ion_descr iptor.sub _segment_num . The value must be between 0 and 256, inclusive.

Field	Type	Default value	Description
sub_segments_expected	integer	null	This is written to segmentation_descriptor.sub_segments_expected. The value must be between 0 and 256, inclusive.

The following table shows the values that MediaTailor automatically sets for some of the segmentation_descriptor message's fields.

Values set by MediaTailor for a segmentation_descriptor message's fields

Field	Type	Value
segmentation_event_cancel_indicator	Boolean	True
program_segmentation_flag	Boolean	True
delivery_not_restricted_flag	Boolean	True

MediaTailor always sets the segmentation_duration_flag to True. MediaTailor populates the segmentation_duration field with the duration, in ticks, of the state content.

Note

When MediaTailor sends the time_signal messages, it sets the splice_command_type field in the splice_info_section message to 6 (0x06).

In HLS output, for an AdBreak with a `time_signal` message, the output `EXT-X-DATERANGE` tag includes a SCTE-35 field that is set to the serialized version of the `splice_info_section` message. For example, the following `EXT-X-DATERANGE` tag shows the serialized version of the `splice_info_section` message:

```
#EXT-X-DATERANGE:ID="\1",START-DATE="\2020-09-25T02:13:20Z",DURATION=300.0,SCTE35-OUT=0xFC002C0000000000000000FFF0050680000000000160214435545490000000100E000019BFCC00E00300000000
```

In DASH output, for an AdBreak with a `time_signal` message, the output `EventStream` element includes a `scte35:SpliceInfoSection` element with `scte35:TimeSignal` and `scte35:SegmentationDescriptor` elements as its children. The `scte35:TimeSignal` element has a child `scte35:SpliceTime` element, and the `scte35:SegmentationDescriptor` element has a child `scte35:SegmentationUpid` element. For example, the following DASH output shows the `EventStream` element's child element structure:

```
<EventStream schemeIdUri="urn:scte:scte35:2013:xml" timescale="90000">
  <Event duration="27000000">
    <scte35:SpliceInfoSection protocolVersion="0" ptsAdjustment="0" tier="4095">
      <scte35:TimeSignal>
        <scte35:SpliceTime ptsTime="0" />
      </scte35:TimeSignal>
      <scte35:SegmentationDescriptor segmentNum="0"
segmentationDuration="27000000" segmentationEventCancelIndicator="false"
segmentationEventId="1" segmentationTypeId="48" segmentsExpected="0">
        <scte35:SegmentationUpid segmentationUpidFormat="hexBinary"
segmentationUpidType="14">012345</scte35:SegmentationUpid>
      </scte35:SegmentationDescriptor>
    </scte35:SpliceInfoSection>
  </Event>
</EventStream>
```

You learned about using SCTE-35 messages to set up ad breaks in channel assembly, the structure and required fields for those messages, and sample HLS and DASH output that includes the SCTE-35 messages.

Time-shifting a channel's playback

With MediaTailor channel assembly, you can time-shift a channel's playback. Time-shifting allows viewers to watch a channel's content at a time other than the original broadcast time. For example,

a viewer can start watching a program from the beginning, even if they join the broadcast after it has started. Or, a viewer can pause a program and resume watching it later.

To time-shift a channel's playback, you add a `start` parameter to the channel's playback URL. The `start` parameter specifies the time when playback should begin, relative to the current time. For example, if the current time is 2:00 PM, and you want to start playback from 1:00 PM, you would set the `start` parameter to `-3600`, which is -1 hour in seconds.

The `start` parameter accepts both positive and negative values, in seconds:

- Negative values indicate a time in the past, relative to the current time. For example, `start=-3600` means "start playback from 1 hour ago."
- Positive values indicate a time in the future, relative to the current time. For example, `start=3600` means "start playback from 1 hour in the future."

The following example shows how to add the `start` parameter to a channel's playback URL:

```
https://a1b2c3d4e5f6.mediapackage.us-west-2.amazonaws.com/out/v1/examplechannel/  
index.m3u8?start=-3600
```

In this example, playback starts from 1 hour ago.

The `start` parameter is subject to the following limitations:

- The maximum time-shift window is 24 hours in the past or future.
- Time-shifting is only available for channels that use the linear playback mode.
- Time-shifting is not available for channels that use the loop playback mode.

If you specify a `start` parameter that is outside the available time-shift window, MediaTailor returns an error.

Using a CDN to optimize MediaTailor ad personalization and content delivery

AWS Elemental MediaTailor works effectively as a standalone service, but integrating it with a content delivery network (CDN), such as Amazon CloudFront or other third-party CDNs, can significantly enhance your streaming workflows. A CDN integration is particularly valuable when you need to serve content to a large, geographically distributed audience or when you want to ensure consistent ad delivery across different AWS Regions.

Without a CDN, your viewers connect directly to MediaTailor for personalized manifests and ad segments, which can lead to increased latency, especially for viewers located far from the AWS Region where your MediaTailor configuration is deployed. Additionally, during high-traffic events, direct connections to MediaTailor might experience increased load, potentially affecting performance.

For more information about MediaTailor concepts and workflows, see [What is AWS Elemental MediaTailor?](#).

When integrating a CDN with MediaTailor, it's important to configure proper CORS (Cross-Origin Resource Sharing) handling to prevent issues that which can cause playback failures in web-based players. Proper CORS configuration is essential for both ad segments and content segments. While ad segments are more susceptible to CORS issues, applying consistent CORS handling across all segment types ensures the most reliable playback experience. For detailed guidance on configuring CDN routing behaviors with proper CORS handling, see [Production-ready CloudFront configuration for MediaTailor](#).

CDN integration also enables advanced parameter passing and dynamic routing capabilities. For information about passing query parameters through CDNs for authorization and routing, see [MediaTailor manifest query parameters](#). For dynamic ad server and origin routing using configuration aliases, see [MediaTailor dynamic ad variables for ADS requests](#).

Placing a CDN between your viewers and MediaTailor provides the following benefits:

- Reduce latency by serving content from edge locations closer to viewers
- Improve scalability by distributing load across the CDN's global infrastructure
- Enhance reliability through redundant delivery paths

- Optimize costs by reducing origin traffic
- Implement advanced features like Media Quality-Aware Routing (MQAR) for improved streaming quality

Topics

- [Select the right CDN for your needs](#)
- [Plan your CDN integration for AWS Elemental MediaTailor](#)
- [Set up CDN integration with MediaTailor](#)
- [Set up SSAI with a CDN for personalized video advertising](#)
- [Build MediaTailor linear channels with channel assembly and CDN](#)
- [Integrate MediaTailor with MediaPackage and CDN](#)
- [Integrating AWS Elemental MediaTailor with Amazon CloudFront](#)
- [Set up third-party CDNs for MediaTailor ad delivery](#)
- [Performance optimization guide for CDN and MediaTailor integrations](#)
- [Monitor MediaTailor CDN operations and performance](#)
- [Testing and validation for CDN and MediaTailor integrations](#)
- [Troubleshoot issues with MediaTailor and CDN integration](#)
- [CDN integration log analysis and error code reference for MediaTailor](#)
- [Automate MediaTailor and CDN with CloudFormation](#)
- [Production-ready CloudFront configuration for MediaTailor](#)
- [Get support and troubleshooting help for CDN and MediaTailor integrations](#)

Select the right CDN for your needs

Choosing the right content delivery network (CDN) provider is an important decision that can impact your content delivery performance, cost, and viewer experience with AWS Elemental MediaTailor. Consider these factors when selecting a CDN for your MediaTailor implementation:

Geographic coverage

Choose a CDN with strong presence in regions where your audience is located. Different CDN providers have varying strengths in different geographic regions.

Integration with AWS services

Amazon CloudFront offers the tightest integration with MediaTailor and other AWS services, which can simplify setup and management. Third-party CDNs might offer other advantages like specialized video delivery features or stronger presence in specific regions.

Video-specific features

Look for CDNs that offer features specifically designed for video delivery, such as adaptive bitrate optimization, video compression, and analytics focused on viewer experience.

Cost structure

Compare pricing models across providers, considering factors like traffic volume, geographic distribution, and feature requirements. Some CDNs offer volume discounts or committed use discounts that might align with your usage patterns.

Support for advanced features

Verify that your chosen CDN supports the features you need, such as token authentication, geo-restriction, request collapsing, and proper header forwarding.

For more information about specific CDN providers and their integration with MediaTailor, see the following resources.

- [CloudFront integration](#) for Amazon CloudFront
- [Third-party CDN setup](#) for third-party CDN providers

The following topics provide comprehensive guidance on configuring MediaTailor with a CDN for optimal performance.

Plan your CDN integration for AWS Elemental MediaTailor

You can improve the viewer experience and reduce latency with a CDN integration for AWS Elemental MediaTailor. When you implement a content delivery network (CDN), you can deliver content from locations that are closer to your viewers. This ensures faster load times, better scalability, and consistent ad delivery across different geographic regions.

You need proper planning before implementing a CDN with AWS Elemental MediaTailor. This section guides you through the key planning areas. You address these areas before beginning the actual configuration. These steps help you create an optimal viewing experience for your audience.

After you complete your planning, see [Integrating a CDN with MediaTailor](#) for step-by-step implementation instructions.

For information about MediaTailor quotas that may affect your CDN planning, see [Quotas in AWS Elemental MediaTailor](#). For information about CloudFront quotas, see [Quotas](#) in the CloudFront Developer Guide.

Before you start planning, understand how MediaTailor interacts with a CDN:

1. The viewers request content through your CDN instead of directly from MediaTailor
2. The CDN forwards manifest requests to MediaTailor for personalization.
3. The CDN caches and serves content segments and ad segments from edge locations.

This architecture reduces load on MediaTailor while you ensure viewers receive personalized ads with minimal latency.

Understanding the manifest terminology helps you configure your CDN correctly. Different streaming protocols use specific manifest structures that affect how you set up the caching and routing:

- *HLS manifests* - When you work with HLS streams, you handle:
 - *Multivariant playlist*: Configure your CDN to route these top-level manifests to MediaTailor for personalization.
 - *Media playlist*: Set appropriate caching rules for these manifests that contain links to content segments.
- *DASH manifests* - When you work with DASH streams, you handle:
 - *MPD (Media Presentation Description)*: Configure your CDN to handle these manifests according to your personalization requirements.

The CDN planning process involves these key steps, each focused on a specific task:

- [Estimate traffic requirements for CDN and MediaTailor integrations](#): Calculate your expected viewer concurrency and the bandwidth requirements.
- [Configure optimization strategies for CDN and MediaTailor integrations](#): Configure your CDN for optimal content delivery and ad personalization.
- [Customize planning for CDN and MediaTailor integrations](#): Adjust your CDN strategy based on your specific MediaTailor workflow.

- [Set up monitoring and scaling for CDN and MediaTailor integrations](#): Implement monitoring and scaling strategies for reliable performance.
- [Optimize costs for CDN and MediaTailor integrations](#): Balance the performance with the cost efficiency.
- [Test your implementation for CDN and MediaTailor integrations](#): Thoroughly test your CDN integration before production deployment.

Estimate traffic requirements for CDN and MediaTailor integrations

To accurately size your content delivery network (CDN) integration with AWS Elemental MediaTailor:

1. Calculate your expected viewer concurrency using historical data or similar events. Plan for additional capacity beyond your baseline to handle unexpected spikes. For current scaling recommendations, consult with your AWS account team. You can also see [Quotas in AWS Elemental MediaTailor](#).
2. Identify peak traffic patterns and potential spikes in your content schedule. Consider factors like:
 - Live sports events or season premieres
 - Marketing campaigns or promotional events
 - Time zone differences for global audiences
 - Holiday or seasonal viewing patterns
3. Determine your bandwidth requirements by multiplying viewer counts by stream bitrates. Work with your CDN provider to calculate appropriate capacity. Base this calculation on your specific content bitrates and expected audience size. Add overhead for ad segments and manifest requests as your provider recommends.
4. Work with your CDN provider to ensure sufficient edge capacity in your target regions.

Ensure your ad insertion capacity meets viewer demand by taking these specific actions:

1. Check your current ad insertion requests quota in the [Service Quotas console](#). Review the current service limits to understand how many concurrent viewers your configuration can support.
2. For high-traffic events, request an increased quota through the [Service Quotas console](#).

3. If you expect more than 500,000 concurrent viewers, then contact [AWS Support](#) at least 2 weeks before your event. This allows AWS to ensure sufficient capacity for your ad personalization needs.

For more information about implementing capacity planning in your workflow, see [Using prefetch scheduling](#) to optimize ad delivery for high-traffic events.

Configure optimization strategies for CDN and MediaTailor integrations

When you complete your traffic estimation, configure your content delivery network (CDN) to optimize content delivery and ad personalization with AWS Elemental MediaTailor. These optimizations help ensure smooth playback while maintaining targeted advertising.

Implement these specific CDN optimizations that follow:

1. Configure origin shield capabilities in your CDN to reduce load on MediaTailor and improve caching efficiency. Origin shield acts as an intermediary caching layer that:
 - Consolidate multiple viewer requests into a single origin request
 - Reduce the number of redundant requests to MediaTailor
 - Improve the response times for the cached content

For implementation details on setting up origin shield with CloudFront, see [Using Origin Shield](#) in the CloudFront Developer Guide.

2. Set appropriate Time To Live (TTL) values for different content types. TTL determines how long the CDN caches content. After this time, the CDN requests a fresh copy from the origin:
 - Manifests:
 - 0 seconds for ad insertion
 - 5-10 seconds for channel assembly

In ad insertion, MediaTailor provides manifests with ads personalized to the viewer. If a playlist or MPD is cached and served to the wrong playback device, the device could encounter playback or tracking issues.

- Content segments: 24 or more hours (these rarely change and you can cache them aggressively to reduce origin load)
- Ad segments: 24 or more hours (ad content is typically reused across viewers and you can cache it for extended periods)

For comprehensive TTL recommendations and caching optimization strategies across all MediaTailor workflows, see [Caching optimization for CDN and MediaTailor integrations](#).

For detailed instructions on configuring cache behaviors in CloudFront, see [Cache Behavior Settings](#) in the CloudFront Developer Guide.

3. Deploy CDN edge nodes close to your viewer populations. Work with your CDN provider to:

- Identify optimal edge node locations based on viewer demographics
- Ensure sufficient capacity in each region
- Monitor edge performance and adjust as needed

For implementation guidance, see [CloudFront edge locations](#) to identify available edge locations for your audience regions.

4. For global audiences, consider implementing a multi-CDN strategy. This approach:

- Uses multiple CDN providers to improve reliability
- Routes viewers to the best-performing CDN for their location
- Provides failover options during CDN outages
- Can optimize costs by leveraging different pricing models

For implementation details, see [Multi-CDN strategies](#) on the AWS Networking & Content Delivery Blog.

Customize planning for CDN and MediaTailor integrations

Different AWS Elemental MediaTailor workflows have unique requirements that affect content delivery network (CDN) planning. For workflow-specific guidance, see [Working with configurations](#).

Adjust your capacity plan based on your specific MediaTailor workflow:

For MediaTailor ad insertion workflows

1. Configure your CDN to handle personalized manifests with zero caching. This ensures each viewer receives unique, targeted ads. The ads are based on their profile and viewing context.
2. Size your ad decision server (ADS) to handle peak request volumes. For guidance on ADS configuration, see [Ad insertion with MediaTailor](#). Consider:
 - Response time requirements for your use case

- Expected concurrent viewer capacity
 - Redundancy and failover requirements
 - Geographic distribution needs
3. Implement request collapsing at the CDN level to handle synchronized ad break requests. Request collapsing combines multiple identical requests into a single origin request. This is crucial during:
- Live sports events when many viewers hit ad breaks simultaneously
 - Popular TV show premieres with synchronized commercial breaks
 - Breaking news events that attract sudden viewer spikes

For implementation details, see [Origin failover](#) to configure request handling during peak loads.

For MediaTailor channel assembly workflows

1. Calculate capacity requirements based on the number of channels and their bitrates. For guidance on channel assembly capacity planning, see [Channel assembly in MediaTailor](#). Consider:
 - Total number of channels
 - Bitrate requirements per channel
 - Expected concurrent viewer load
 - Geographic distribution needs
2. Configure your CDN to handle predictable traffic patterns based on published schedules. Channel assembly typically has more predictable patterns than ad insertion because:
 - Programming schedules are known in advance
 - Viewer behavior follows established patterns
 - Content doesn't change dynamically per viewer
3. Ensure your origin has sufficient bandwidth to maintain consistent channel output. Implement:
 - Redundant origin servers for high-availability channels
 - Automated failover between primary and backup origins
 - Monitoring to detect origin performance issues

For implementation guidance, see [Setting up origin redundancy](#) to create a resilient origin infrastructure.

For combined MediaTailor workflows

1. Size your infrastructure to handle the combined traffic patterns of both services. For guidance on combined workflows, see [Using AWS Elemental MediaTailor to insert ads](#). Consider:
 - Channel assembly baseline requirements
 - Ad insertion overhead requirements
 - Peak traffic patterns
 - Redundancy needs
2. Configure separate CDN behaviors for linear content delivery and dynamic ad insertion. This separation allows you to:
 - Optimize caching policies for each content type independently
 - Route requests to appropriate origins based on content type
 - Monitor performance metrics separately for each workflow
3. Set up proper routing between edge and origin CDNs to maintain optimal performance. Consider using:
 - Different origin paths for content segments (/content/*) and ad segments (/ads/*)
 - Separate cache behaviors for manifests and segments
 - Geographic routing to optimize latency for different regions

For implementation details, see [Configuring cache behaviors](#) to set up path-based routing and caching rules.

Set up monitoring and scaling for CDN and MediaTailor integrations

Effective monitoring and scaling strategies are crucial for maintaining optimal performance and viewer experience with your AWS Elemental MediaTailor content delivery network (CDN) integration. Implement these approaches to ensure your CDN integration performs reliably at scale.

Implement these monitoring and scaling strategies that follow:

1. Configure monitoring for these key metrics. For guidance on appropriate target values, see [Monitoring MediaTailor with Amazon CloudWatch](#) and consult your CDN provider's best practices:

- CDN cache hit ratios (establish baseline metrics and targets based on your content type and delivery patterns)
- Origin request volumes (monitor patterns during normal operation to establish baselines for anomaly detection)
- Error rates by error type (define thresholds based on your service level objectives and MediaTailor best practices)
- Response times (set appropriate latency targets based on your viewer experience requirements and geographic distribution)

For detailed implementation instructions, see [Creating CloudWatch dashboards](#) to visualize your MediaTailor and CDN metrics together.

2. Set up alerts for unexpected traffic patterns or performance degradation. Configure thresholds based on your baseline metrics and service level objectives. For guidance on setting up alerts, see [Creating Amazon CloudWatch alarms](#). Consider monitoring:
 - Significant deviations from baseline cache hit ratios (typically alert when falling under 85-90%)
 - Sudden increases in origin request volume (alert on 30% or greater increase from baseline)
 - Error rate spikes exceeding your defined thresholds (typically 1-2% for 4xx errors, 0.5% for 5xx errors)
 - Response time degradation beyond acceptable levels (typically >500ms for manifests, >200ms for segments)

For implementation examples, see [CloudWatch concepts](#) for creating effective monitoring dashboards and alerts.

3. Create scaling plans for predictable high-traffic events. Your plans should include these key elements:
 - Pre-event capacity increases (24-48 hours before event start)
 - Gradual viewer ramp-up schedules (typically 10-20% of expected audience per 5-minute interval)
 - Regional capacity distribution based on audience (allocate capacity proportionally to expected regional viewership)
 - Post-event scaling procedures (maintain peak capacity for 30-60 minutes after event conclusion)

For implementation guidance on scaling for high-traffic events, see [Setting up a resilient end-to-end live workflow](#) on the AWS Media Blog.

4. Implement failover and redundancy measures for critical streams, including:

- Multi-region CDN deployments (at least two regions for critical content)
- Backup origin servers (configured with automated health checks every 30 seconds)
- Automated failover triggers based on health checks (typically after 2-3 failed checks)
- Recovery procedures for different failure scenarios (documented with specific response time targets)

For detailed implementation steps, see [Optimizing high availability with CloudFront origin failover](#).

Optimize costs for CDN and MediaTailor integrations

Content delivery network (CDN) costs can vary significantly based on traffic patterns, geographic distribution, and feature usage with AWS Elemental MediaTailor. For current pricing information, see [CloudFront pricing](#). You can also consult your CDN provider's documentation.

Balance the performance with the cost efficiency by using these strategies:

1. Analyze your CDN traffic patterns to select the most cost-effective pricing tier. Review the following factors with your CDN provider:

- Data transfer volumes by region and time period
- Request patterns for manifests and segments
- Geographic distribution requirements
- Peak versus average usage patterns

For help with cost analysis, use the [AWS Pricing Calculator](#) to estimate your CloudFront costs based on your specific usage patterns.

2. For predictable workloads, evaluate reserved capacity agreements with your CDN provider. These agreements can offer benefits such as:

- Discounted rates for committed usage volumes
- Predictable monthly costs for budgeting
- Priority support and capacity allocation

Consult with your CDN provider to determine if reserved capacity is appropriate for your usage patterns. For CloudFront, see [CloudFront premium features](#). This provides information about reserved capacity options.

3. Optimize egress costs by balancing traffic between MediaTailor and your CDN provider. Strategies include:

- Maximize cache hit ratios to reduce origin requests
- Use origin shield to consolidate requests
- Implement compression to reduce data transfer volumes
- Choose CDN regions that align with AWS Region pricing

For implementation guidance on compression, see [Serving compressed files](#) in the CloudFront Developer Guide.

4. Implement appropriate caching strategies for different content types to reduce origin requests. For guidance on cache optimization, see [Improving cache hit ratios](#). Focus on:

- Content segments (you can cache these for extended periods)
- Ad segments (typically reused across viewers)
- Static assets like player files and images

Improved cache hit ratios significantly reduce origin costs. Work with your CDN provider to optimize cache configurations for your specific content patterns. For detailed implementation steps, see [Configuring cache behaviors](#).

Test your implementation for CDN and MediaTailor integrations

Ensure reliable viewer experiences by thoroughly testing your AWS Elemental MediaTailor content delivery network (CDN) integration before production deployment. Proper testing helps identify and resolve issues before they impact your audience. For guidance on testing methodologies, see [Testing CloudFront distributions](#). You can also consult your CDN provider's testing documentation.

Follow these testing steps that validate your CDN integration:

1. Create a test environment that mirrors your production configuration. Include:
 - Identical CDN settings and cache behaviors
 - Include representative content with various bitrates and formats
 - Configure test ad decision server with sample ad responses

- Set up monitoring and alerting configurations

For step-by-step implementation guidance, see [Creating a staging distribution](#) in the CloudFront Developer Guide.

2. Conduct load testing to verify your capacity estimates. For guidance on load testing, see [Monitoring MediaPackage](#). Test scenarios should include:

- Gradual viewer count increases (typically 10-20% of expected peak every 5 minutes)
- Sudden traffic spikes based on your expected peak loads (simulate 50% of peak audience joining within 60 seconds)
- Extended peak load periods (maintain peak load for at least 30-60 minutes)
- Geographic distribution matching your audience (distribute test traffic according to expected viewer locations)

Validate that response times remain under target thresholds. Typically, this means less than 500ms for manifests and less than 200ms for segments. Error rates should stay under 1%. For implementation details on load testing tools and methodologies, see [Load testing with CloudFront](#) on the AWS Networking & Content Delivery Blog.

3. Test failover scenarios to ensure reliability. Simulate:

- Origin server failures (complete outage and partial degradation scenarios)
- CDN edge location outages (test with traffic routing to backup locations)
- Ad decision server unavailability (test with 5-10 second timeouts)
- Network connectivity issues (simulate packet loss and latency increases)

Work with your CDN provider to establish appropriate failover response time targets for your use case. Typically, this means less than 3 seconds for failover completion. For implementation guidance on failover testing, see [Origin failover](#) in the CloudFront Developer Guide.

4. For major events, implement gradual ramp-up strategies based on [AWS load testing guidelines](#):

- Stagger start times for different audience segments. For example, allow premium subscribers first with 15-minute intervals between audience groups.
- Use pre-warming techniques to gradually increase load. Pre-warming involves:
 - Populate CDN caches with popular content 2-4 hours before the event
 - Gradually increase synthetic traffic to 20-30% of expected peak to warm up systems
 - Test all components under realistic load conditions with actual content

- Monitor system performance throughout the ramp-up period, tracking:

- Cache hit ratios and response times (target >90% hit ratio, <500ms response)
- Error rates and origin load (maintain error rates <1%, origin CPU <70%)
- Ad personalization success rates (target >98% successful personalization)
- Viewer experience metrics (target <2 second startup time, <0.5% rebuffering)
- Have a contingency plan for unexpected traffic surges. Your plan should include these essential components:
 - Emergency capacity scaling procedures with documented steps to increase capacity by 50-100% within 15 minutes
 - Backup CDN activation protocols with ability to shift 20-50% of traffic to secondary CDN
 - Simplified ad insertion fallback to reduce targeting parameters from 10 or more to 3-5 essential ones
 - Communication plans for stakeholders with pre-defined notification templates and contact lists

After completing your testing, proceed to [Implementing your CDN integration](#) for production deployment steps.

Troubleshoot common issues for CDN and MediaTailor integrations

Address common content delivery network (CDN) integration challenges with AWS Elemental MediaTailor before they impact your viewers. This section helps you identify and resolve typical issues that occur during CDN integration with AWS Elemental MediaTailor.

For comprehensive troubleshooting guidance, see [Troubleshooting MediaTailor](#) and [Troubleshooting CloudFront distributions](#).

Resolving MediaTailor manifest delivery problems

If viewers experience playback issues or see incorrect ads, check these common manifest-related problems:

- **Incorrect caching settings:** If your CDN caches personalized manifests, viewers may see ads intended for other users.

Solution: Configure your CDN with a cache TTL of 0 for manifest requests to MediaTailor.

- **Origin request failures:** If your CDN can't reach MediaTailor, then manifest requests will fail.

Solution: Check network connectivity between your CDN and MediaTailor. Verify your CDN properly forwards the correct headers.

- **Session parameter issues:** Missing or incorrect session parameters can cause personalization failures.

Solution: Ensure that your player is correctly appending all required session parameters to the manifest requests.

Fixing MediaTailor segment delivery problems

If content or ad segments aren't loading properly, investigate these common issues:

- **Segment path rewriting:** Incorrect CDN configuration may rewrite segment URLs improperly.

Solution: Verify your CDN correctly handles segment URLs. Ensure it doesn't modify paths in a way that breaks references.

- **CORS configuration:** Missing or incorrect CORS headers can prevent browsers from loading segments.

Solution: Configure your CDN to pass the appropriate CORS headers for segment requests.

- **Cache miss storms:** During high-traffic events, multiple cache misses can overwhelm origin servers.

Solution: Implement request collapsing and origin shield capabilities to reduce origin load during traffic spikes.

Addressing MediaTailor CDN performance problems

If viewers experience buffering or slow loading, check these performance-related issues:

- **Low cache hit ratio:** If your CDN frequently requests content from the origin, then performance will suffer.

Solution: Analyze the cache hit ratios by content type and adjust the TTL settings to improve caching efficiency.

- **Geographic distribution:** Viewers distant from CDN edge locations may experience increased latency.

Solution: Review your CDN edge location distribution. Add capacity in regions with high viewer concentrations.

- **Origin capacity limitations:** If your origin servers are overloaded, then response times will increase.

Solution: Implement origin request limiting. You can also increase origin capacity or improve caching to reduce origin load.

For additional troubleshooting assistance, see [Troubleshooting MediaTailor](#) or contact AWS Support. For implementation guidance on resolving common CDN issues, see [Debugging your content delivery network](#) on the AWS Media Blog.

Set up CDN integration with MediaTailor

This section provides guidance on integrating AWS Elemental MediaTailor with a content delivery network (CDN).

Effective CDN integration with MediaTailor is essential for delivering high-quality streaming experiences with personalized ads at scale. This guide walks you through the complete process of setting up, configuring, and optimizing your CDN integration.

For additional information, see the following links:

- For information about passing query parameters through CDNs for authorization and ad targeting, see [MediaTailor manifest query parameters](#).
- For advanced routing using dynamic variables and configuration aliases, see [MediaTailor dynamic ad variables for ADS requests](#).
- For information about creating MediaTailor configurations, see [Using AWS Elemental MediaTailor to insert ads](#).
- For information about creating a CloudFront distribution, see [Creating a distribution](#) in the CloudFront Developer Guide.

To ensure clarity in this guide, the following terminology is used to describe different types of manifests:

- **HLS manifests:**

- *Multivariant playlist*: The top-level manifest that contains links to media playlists
- *Media playlist*: The second-level manifest with links to content segments
- **DASH manifests**:
 - *MPD (Media Presentation Description)*: The standard term for DASH manifests

Integrating MediaTailor with a CDN provides the following benefits:

- Reduced latency for viewers
- Improved scalability for high-traffic events
- Enhanced reliability through redundant delivery paths
- Optimized costs by reducing origin traffic
- Better protection against DDoS attacks

CDN integration components and requirements

A successful CDN integration with MediaTailor involves configuring the following key components:

CDN routing behaviors

Rules that determine how different types of requests (manifests, content segments, ad segments) are routed through your CDN.

CDN mapping in MediaTailor

Configuration in MediaTailor that ensures manifests reference your CDN domain instead of directly referencing origin servers.

Security settings

Configurations that protect your content and infrastructure, including transport security, access control, and monitoring.

Testing and validation

Procedures to verify that your CDN integration is working correctly before deploying to production.

Prerequisites for CDN integration

Before configuring your CDN integration, ensure you have the following:

1. A MediaTailor configuration with the following settings:

- Your content origin as the **Content source**
- Your ADS as the **Ad decision server**

You need the origin and ADS URLs in the CDN integration steps as well.

2. Access to your CDN's configuration interface

3. Understanding of your CDN's specific terminology for behaviors, rules, and cache settings

4. Knowledge of your content structure, including file extensions used for segments (such as .ts, .mp4, or .m4s)

CDN integration setup steps

The process of integrating MediaTailor with a CDN follows these high-level steps:

1. **Configure CDN routing behaviors** - Set up your CDN to route different types of requests appropriately.
2. **Configure CDN mapping in MediaTailor** - Update your MediaTailor configuration to use your CDN domain names.
3. **Implement security best practices** - Configure security settings to protect your content and infrastructure.
4. **Test your integration** - Verify that your CDN integration is working correctly.

Required headers for MediaTailor CDN integration

For MediaTailor to function correctly with your CDN, you must configure your CDN to forward specific HTTP headers. These headers are essential for proper functionality including compression, device detection, ad personalization, and geo-targeting.

Configure your CDN to forward the following headers to MediaTailor:

Accept-Encoding

Purpose: Required for compression functionality

Details: This header tells MediaTailor which compression methods the client supports. MediaTailor uses this information to compress manifests when possible, reducing bandwidth usage and improving performance. Legacy devices that don't support compression won't send this header, and MediaTailor will return uncompressed manifests.

User-Agent

Purpose: Required for device detection and ad personalization

Details: MediaTailor analyzes the User-Agent header to identify the client device type, browser, and capabilities. This information is used for ad targeting, device-specific optimizations, and ensuring compatibility with different playback clients.

Host

Purpose: Required for proper request routing

Details: The Host header ensures that requests are routed to the correct MediaTailor endpoint. This is particularly important in multi-tenant environments and when using custom domain configurations.

Many CDNs, including Amazon CloudFront, do not forward the Host header by default. For CloudFront users: See [Configuring cache behaviors](#) for configuration instructions.

X-Forwarded-For

Purpose: Required for client IP detection and geo-targeting

Details: This header preserves the original client IP address when requests pass through your CDN. MediaTailor uses this information for geographic ad targeting, analytics, and compliance with regional content restrictions.

Important

All four headers are required for full MediaTailor functionality. Missing any of these headers can result in reduced functionality, including:

- Inability to compress manifests (missing Accept-Encoding)
- Poor ad targeting and device compatibility issues (missing User-Agent)
- Request routing failures (missing Host)
- Inaccurate geo-targeting and analytics (missing X-Forwarded-For)

For CDN-specific configuration instructions, see the routing behaviors and caching sections that reference this header list.

The following topics provide detailed instructions for each aspect of CDN integration.

Topics

- [Set up CDN routing behaviors for MediaTailor](#)
- [Set up CDN mapping in MediaTailor](#)
- [CDN integration security best practices for MediaTailor](#)

Set up CDN routing behaviors for MediaTailor

This section explains how to set up your content delivery network (CDN) to route different types of requests appropriately for AWS Elemental MediaTailor integration. Proper routing configuration ensures that manifest requests, content segments, and ad segments are handled correctly.

Configuring CDN routing behaviors is a critical step in creating an efficient content delivery pipeline. By setting up specific routing rules for different content types, you can optimize caching, improve delivery performance, and ensure personalized ad insertion works correctly.

For advanced routing scenarios using dynamic variables and configuration aliases, see [MediaTailor dynamic ad variables for ADS requests](#). For information about preserving query parameters across CDN routing, see [MediaTailor manifest query parameters](#).

Important

Failure to include CORS headers in either the cached object or on the CDN response to viewers can cause playback failures.

CDN routing behavior configuration

Set up your CDN to route different types of requests appropriately.

Content segment routing

Content segment routing directs requests for your actual content segments to your origin server. Like ad segment routing, content segment routing also requires proper CORS configuration to ensure smooth playback in web-based players.

For detailed configuration guidance, see the CloudFront example at [Precedence 4: Content origin path behavior](#). This example provides specific settings that you should follow for CloudFront or adapt for other CDNs.

Key configuration requirements for content segment routing include:

- Use path patterns that match your content segment file extensions (like *.ts, *.mp4, or *.m4s)
- Route requests to your content origin (such as an Amazon S3 Bucket or MediaPackage endpoint)
- For optimal cache-hit ratio, include only query string parameters that cause your origin to modify the response in the cache key and forward the origin request
- Apply an appropriate cache policy with greater than 24-hour TTL values
- Include CORS response headers to your viewers

Ad segment routing

Ad segment routing is critical for delivering personalized advertisements to viewers. When configuring ad segment routing, you must implement proper CORS (Cross-Origin Resource Sharing) handling to prevent issues that can cause playback failures in web-based players.

For detailed configuration guidance, see the CloudFront example at [Precedence 0: Ad segments path behavior](#). This example provides specific settings that you should follow for CloudFront or adapt for other CDNs.

Key configuration requirements for ad segment routing include:

- Use the path pattern /tm/* specifically for MediaTailor ad segments
- Route requests to segments.mediatailor.*region*.amazonaws.com
- For optimal cache-hit ratio, do not include any viewer request headers, cookies, or query string parameters in the cache key or in the origin request
- Apply an appropriate cache policy with greater than 24-hour TTL values
- Include CORS response headers to your viewers

Manifest request routing

To route multivariant playlist, media playlist, and MPD requests to MediaTailor, use the following general settings. For CloudFront configuration, see [Configuring manifest cache behavior](#).

1. In your CDN configuration interface, create behaviors for different manifest types.
2. Set path patterns to match multivariant playlist and media playlist file extensions (* .m3u8 for HLS) and MPD file extensions (* .mpd for DASH).
3. Configure the origin setting in your CDN to point to your MediaTailor configuration endpoint.
4. For ad insertion, disable caching of personalized multivariant playlists, media playlists, and MPDs. Because ad insertion provides personalized manifests, your CDN shouldn't cache them. If a different playback device than the one intended receives a cached playlist or MPD, it could result in issues with playback or tracking. For comprehensive caching guidance including TTL recommendations for all content types, see [Caching optimization for CDN and MediaTailor integrations](#).
5. Configure header forwarding for all headers. For minimum requirements, see [Required headers for MediaTailor CDN integration](#).
6. Enable query string forwarding to pass ad targeting parameters.

HLS multivariant playlist

HLS multivariant playlist requests follow these formats:

```
https://<playback-endpoint>/v1/index/<hashed-account-id>/<origin-id>/<index>.m3u8
```

Example:

```
https://777788889999.mediatailor.us-east-1.amazonaws.com/v1/master/  
a1bc06b59e9a570b3b6b886a763d15814a86f0bb/Demo/assetId.m3u8
```

HLS media playlist

HLS media playlist requests follow these formats:

```
https://<playback-endpoint>/v1/manifest/<hashed-account-id>/<session-id>/  
<manifestNumber>.m3u8
```

Player requests to `https://CDN_Hostname/some/path/asset.m3u8` are routed to the MediaTailor path `https://mediatailor.us-west-2.amazonaws.com/v1/session/configuration/endpoint` based on the keyword `*.m3u8` in the request.

Example:

```
https://777788889999.mediatailor.us-east-1.amazonaws.com/v1/manifest/  
a1bc06b59e9a570b3b6b886a763d15814a86f0bb/c240ea66-9b07-4770-8ef9-7d16d916b407/0.m3u8
```

DASH MPD

DASH MPD requests follow these formats:

```
https://<playback-endpoint>/v1/dash/<hashed-account-id>/<origin-id>/<assetName>.mpd
```

Player requests to `https://CDN_Hostname/some/path/asset.mpd` are routed to the MediaTailor path `https://mediatailor.us-west-2.amazonaws.com/v1/dash/configuration/endpoint` based on the keyword `*.mpd` in the request.

Example:

```
https://777788889999.mediatailor.us-east-1.amazonaws.com/v1/dash/  
a1bc06b59e9a570b3b6b886a763d15814a86f0bb/Demo/0.mpd
```

CDN routing best practices

When configuring CDN routing behaviors, follow these best practices to ensure optimal performance and reliability:

Use specific path patterns

Create specific path patterns that accurately match your content structure to ensure proper routing.

Prioritize behavior order

In most CDNs, behaviors are evaluated in order. Place more specific behaviors before more general ones.

Test behavior patterns

Verify that your path patterns correctly match the expected requests before deploying to production.

Document your configuration

Maintain documentation of your CDN routing behaviors to facilitate troubleshooting and future updates.

Next steps

After configuring your CDN routing behaviors, the next step is to configure CDN mapping in MediaTailor. See [Set up CDN mapping in MediaTailor](#) for instructions.

Set up CDN mapping in MediaTailor

This section explains how to configure AWS Elemental MediaTailor to use your content delivery network (CDN) domain names. After setting up your CDN routing behaviors, you need to update your MediaTailor configuration to ensure that manifests reference your CDN domain instead of directly referencing origin servers.

Configuring CDN mapping in MediaTailor ensures that all content and ad segment URLs in your manifests point to your CDN rather than directly to origin servers. This step is essential for creating a complete CDN delivery chain and maximizing the benefits of your CDN integration.

CDN mapping configuration in MediaTailor

After setting up your CDN routing behaviors, configure MediaTailor to use your CDN domain names:

1. Open the [MediaTailor console](#).
2. Select the configuration you want to update.
3. In the **Advanced settings** of the **CDN configuration** section, enter your CDN domain name in the **CDN content segment prefix** field.
4. If you're using a separate CDN domain for ad segments, enter it in the **CDN ad segment prefix** field.
5. Save your changes.

This configuration ensures that MediaTailor generates manifests with URLs that point to your CDN instead of directly to the origin or ad segment storage.

Understanding base URL behavior

MediaTailor determines base URLs in manifests based on your CDN prefix configuration:

- **CDN ad segment prefix configured:** Ad segments use the CDN prefix as the base URL.
- **CDN ad segment prefix not configured:** Ad segments use the direct MediaTailor location as the base URL.

- **CDN content segment prefix configured:** Content segments use the CDN prefix as the base URL.
- **CDN content segment prefix not configured:** Content segments reference the original content origin.

DASH BaseURL handling

For DASH manifests, MediaTailor manages BaseURL settings differently for content and ad segments:

Content segments:

- **With CDN content segment prefix:** MediaTailor sets exactly one BaseURL at the MPD level using your specified prefix.
- **Without CDN content segment prefix:** MediaTailor preserves existing BaseURL settings from the origin manifest, or adds one based on the origin MPD URL if none exist.

Ad segments:

- **With CDN ad segment prefix:** Each ad period gets exactly one BaseURL using the configured prefix.
- **Without CDN ad segment prefix:** Each ad period gets exactly one BaseURL pointing to the MediaTailor ad content server.

Example CDN mapping example

If your content origin is `http://origin.com/contentpath/` and your CDN content segment prefix is `https://cdn.example.com/`, then a content segment that would normally be referenced as `http://origin.com/contentpath/subdir/content.ts` will appear in the manifest as `https://cdn.example.com/subdir/content.ts`.

Important considerations

When configuring CDN mapping in MediaTailor, keep the following important considerations in mind:

Use HTTPS for CDN prefixes

Always use HTTPS URLs for your CDN prefixes to ensure secure content delivery.

Match CDN behavior paths

Ensure that the CDN prefixes you configure in MediaTailor match the path patterns you configured in your CDN behaviors.

Consider regional CDNs

If you're using different CDN domains for different regions, you'll need to create separate MediaTailor configurations for each region.

Verify domain ownership

Ensure that you have control over the CDN domains you configure in MediaTailor.

Verify CDN mapping configuration

After configuring CDN mapping in MediaTailor, verify that your configuration is working correctly:

1. Request a manifest through your CDN.
2. Examine the manifest content to verify that segment URLs reference your CDN domain.
3. Check that content segment URLs in the manifest point to your CDN domain.
4. Check that ad segment URLs in the manifest point to your CDN domain.

For comprehensive testing and validation procedures, see [Testing and validation for CDN and MediaTailor integrations](#).

Next steps

After configuring CDN mapping in MediaTailor, the next step is to implement security best practices for your CDN integration. See [CDN integration security best practices for MediaTailor](#) for instructions.

CDN integration security best practices for MediaTailor

This section explains how to implement security best practices for your AWS Elemental MediaTailor content delivery network (CDN) integration. Proper security configuration helps protect your content and infrastructure from unauthorized access and potential threats.

Securing your CDN integration is crucial for protecting your content, preventing unauthorized access, and ensuring compliance with security requirements. Implementing these best practices helps create a robust security posture for your streaming workflow.

CDN security configuration

Before configuring your CDN routing and cache settings, implement these security best practices to protect your content and infrastructure:

Transport security

MediaTailor only uses HTTPS for all communication and does not allow HTTP connections. To ensure secure communication between all components in your streaming workflow, complete the following steps:

1. Configure HTTPS for all communication between your CDN and MediaTailor endpoints.
MediaTailor requires HTTPS and will not accept HTTP connections.
2. Use TLS 1.2 or later for all HTTPS connections.
3. Configure your CDN to enforce HTTPS-only connections from viewers.

Access control

To protect your content and infrastructure from unauthorized access, implement the following access controls:

1. Configure geo-restriction settings if you need to limit content access to specific regions.
2. Implement signed URLs or signed cookies for content that requires viewer authentication.
3. Set up IP-based allowlists for administrative access to CDN configuration.

Security monitoring

To detect and respond to security events effectively, implement the following monitoring practices:

1. Enable access logging for your CDN distribution.
2. Set up alerts for unusual traffic patterns or access attempts.
3. Regularly review security configurations and update as needed.

Advanced security features

The following advanced security features provide enhanced protection for your CDN integration:

Web Application Firewall (WAF)

If your CDN supports WAF functionality, configure it to protect against common web vulnerabilities and attacks.

DDoS protection

Enable DDoS protection features provided by your CDN to mitigate distributed denial-of-service attacks.

Origin access control

Configure your origin servers to accept requests only from your CDN to prevent direct access attempts.

Content encryption

For highly sensitive content, consider implementing additional encryption mechanisms beyond transport security.

Next steps

After implementing security best practices, the next step is to test and troubleshoot your CDN integration. See [Troubleshoot CDN integration](#) for comprehensive testing and troubleshooting instructions.

Set up SSAI with a CDN for personalized video advertising

This section provides comprehensive guidance for integrating AWS Elemental MediaTailor server-side ad insertion (SSAI) with a content delivery network (CDN). Follow these steps to set up, configure, and optimize your SSAI CDN integration.

Server-side ad insertion (SSAI) is a technology that seamlessly inserts personalized advertisements into video streams at the server level rather than the client level. When combined with a CDN, you create a robust, scalable solution for delivering personalized advertising to global audiences with minimal latency.

In this topic, we use the term *manifests* to refer collectively to multivariant playlists, media playlists, and MPDs.

What you'll need

Before setting up MediaTailor ad insertion with a CDN, gather these required resources:

AWS account and permissions

An AWS account with appropriate permissions to create and manage MediaTailor resources

IAM permissions for MediaTailor, CloudFront (if using), and related services

For detailed permission requirements, see [Security in AWS Elemental MediaTailor](#).

Required services

AWS Elemental MediaTailor configured and running

A content delivery network (CDN) account (Amazon CloudFront or third-party CDN)

Origin server for your content (HLS or DASH)

Ad decision server (ADS) that supports VAST or VMAP

Content requirements

Content properly encoded and packaged in HLS or DASH format

Ad break markers in your content (for VOD) or SCTE-35 markers (for live)

Before you begin

Complete these setup tasks before implementing MediaTailor ad insertion with a CDN:

1. Configure network connectivity between your CDN, MediaTailor, and origin servers
2. Set up HTTPS for secure content delivery
3. Configure DNS settings for your CDN domain
4. For basic MediaTailor setup, complete the steps in [Setting up](#) and [Getting started with MediaTailor ad insertion](#).

Knowledge prerequisites

To successfully implement this solution, you should have:

- Understanding of streaming protocols (HLS/DASH)
- Basic knowledge of CDN configuration principles
- Familiarity with ad insertion concepts

Benefits of CDN integration

Integrating SSAI with a CDN delivers these key benefits:

Improved viewer experience

Properly configured CDNs reduce buffering, startup times, and playback errors during ad transitions. This results in higher viewer engagement and satisfaction.

Cost reduction

Efficient caching strategies minimize origin requests. This reduces data transfer costs and origin server load, particularly important for high-volume ad-supported content.

Scalability

Optimized CDN configurations handle traffic spikes during popular events without degrading performance, ensuring your personalized ads are delivered even during peak viewing times.

Global reach

Properly configured CDNs deliver content with low latency to viewers worldwide, regardless of their location, expanding your potential audience.

Seamless ad transitions

Optimized CDN configuration ensures smooth transitions between content and ads, creating a broadcast-quality viewing experience.

The following topics provide comprehensive guidance on configuring MediaTailor with a CDN for optimal performance.

Topics

- [Understand ad insertion architecture for CDN and MediaTailor integrations](#)
- [Set up basic MediaTailor SSAI with a CDN for optimal ad delivery](#)
- [Integrate MediaTailor SSAI with channel assembly for monetized linear channels](#)

- [Optimize performance for CDN and MediaTailor integrations](#)
- [Monitor operations for CDN and MediaTailor integrations](#)
- [Troubleshoot MediaTailor SSAI with CDNs for uninterrupted ad delivery](#)

Understand ad insertion architecture for CDN and MediaTailor integrations

This section explains the concepts and architecture of server-side ad insertion (SSAI) with content delivery networks (CDNs) for AWS Elemental MediaTailor. You'll learn how dynamic ad insertion and manifest manipulation work together to enable effective video monetization.

Server-side ad insertion (SSAI) with MediaTailor allows you to:

- Insert personalized advertising into your video streams at defined ad break points
- Target ads precisely based on viewer data
- Eliminate the need for client-side ad insertion technology

When combined with a CDN, you can deliver these personalized streams to viewers with improved performance and scalability, enhancing your video monetization strategy.

The recommended architecture for ad insertion with a CDN positions the CDN between viewers and ad insertion, with ad insertion accessing content directly from your origin. This architecture provides the following benefits for both content delivery and video monetization:

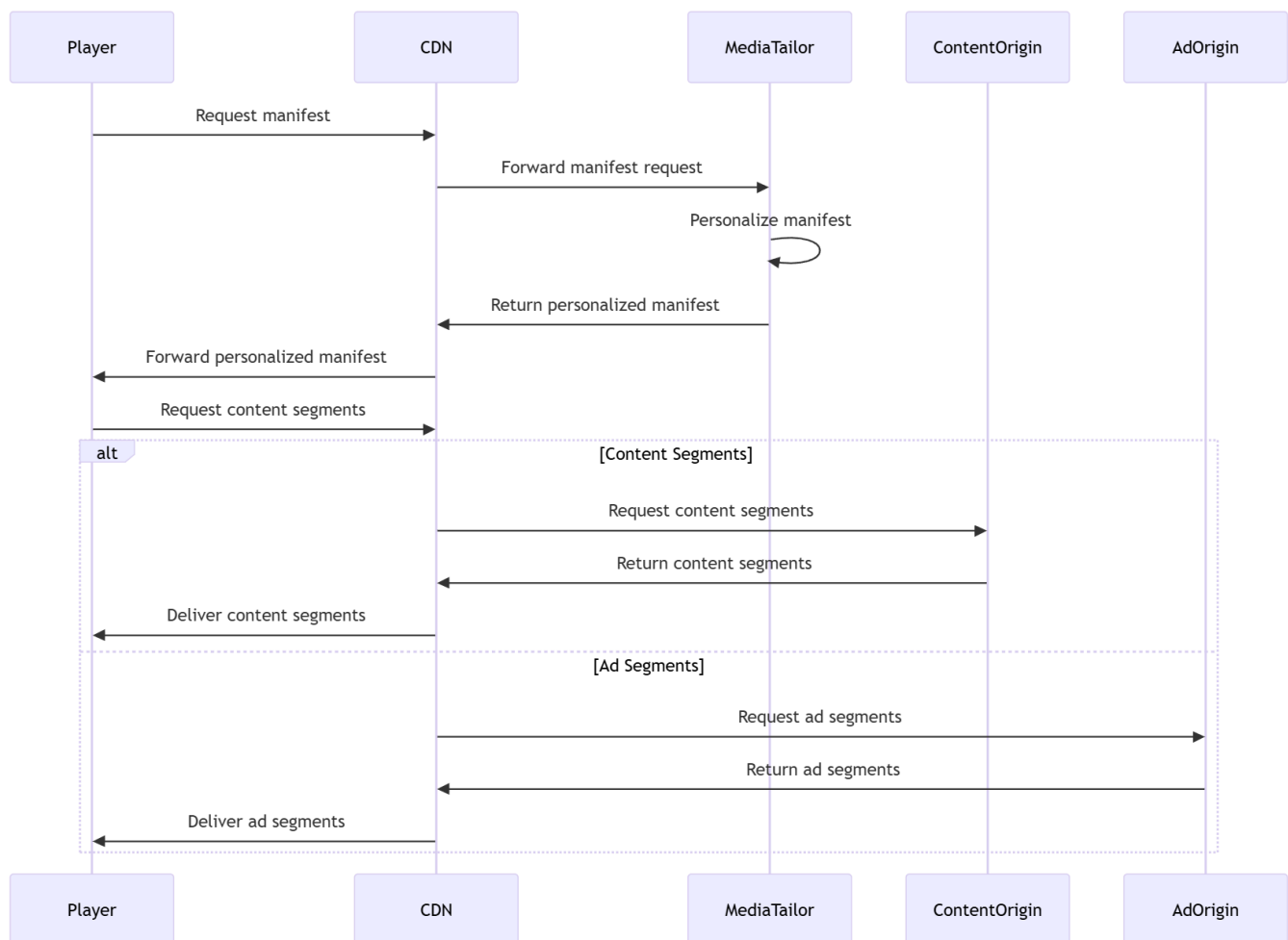
- Effective caching of content and ad segments
- Reduced request load on MediaTailor
- Improved delivery speed to viewers
- Simplified URL management
- Consistent delivery of personalized advertising across devices

In this recommended architecture:

1. Viewers request manifests from the CDN
2. CDN forwards requests to ad insertion

3. Ad insertion requests content manifests from the origin
4. Ad insertion requests ads from the ad decision server (ADS)
5. Ad insertion personalizes manifests by replacing the ad markers (from the origin manifest) with URLs that point to targeted ad segments for the specific viewer (from the ADS)
6. Ad insertion returns the personalized manifests containing ad segment URLs to the CDN, which forwards them to viewers
7. Viewers request segments through the CDN
8. CDN routes segment requests based on the segment type:
 - Content segment requests go to the content origin
 - Ad segment requests go to MediaTailor

This architecture ensures optimal performance while maintaining the security and flexibility benefits of using a CDN.



Note

This flow varies slightly between VOD and live content. For VOD, manifests can be cached longer, while live content requires more frequent manifest updates to maintain stream continuity.

The key difference between VOD and live content caching:

VOD content

Set longer TTL values (minutes to hours) for manifests because they don't change frequently

Live content

Set shorter TTL values (seconds) for manifests to ensure viewers receive the most current stream segments

We don't recommend that you position a CDN between your content origin and AWS Elemental MediaTailor. Doing so can introduce several technical challenges:

Cache-key collisions

Configure your CDN to properly handle query parameters. This prevents MediaTailor from receiving incorrect manifests when requesting the same manifest with different query parameters.

Gzip compression issues

If you experience manifest parsing errors, ensure your CDN delivers properly formatted manifests to MediaTailor. Some CDNs may deliver corrupt gzip payloads that can cause parsing failures. If this occurs, you may need to disable compression between your CDN and MediaTailor while maintaining compression for cost savings elsewhere in your workflow.

Manifest freshness

For live streams, configure your CDN to deliver current manifests to MediaTailor. This prevents synchronization issues between content and ads.

Performance optimization

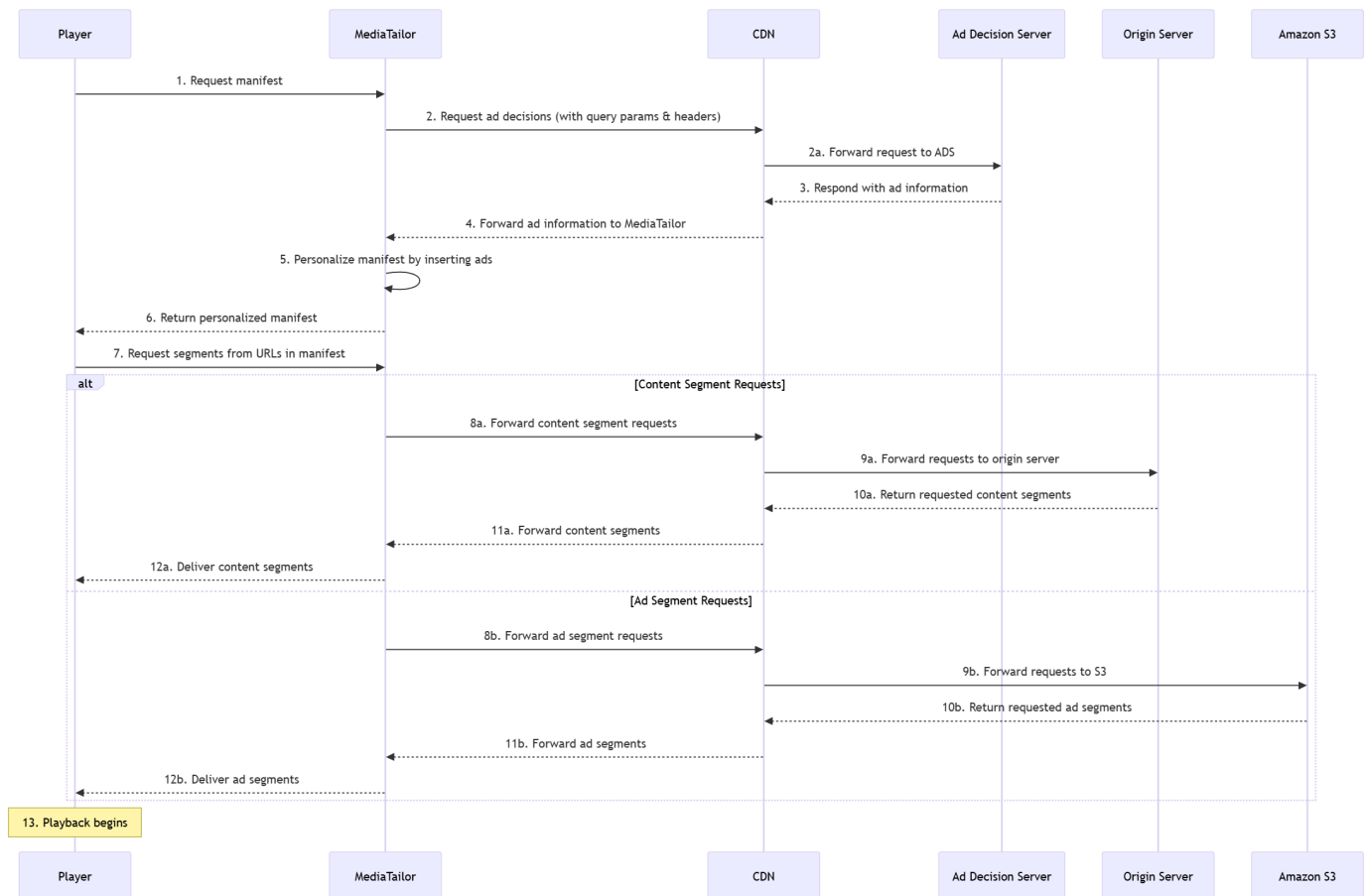
Minimize network hops and potential cache misses to reduce playback start-up times.

Cache management

Implement simplified cache invalidation strategies, especially for live content where manifests update frequently.

In this sub-optimal architecture:

1. Viewers request multivariant playlists, media playlists, or MPDs directly from AWS Elemental MediaTailor.
2. MediaTailor requests content manifests (multivariant playlists, media playlists, or MPDs) through the CDN.
3. The CDN forwards requests to the origin server.
4. The origin server returns multivariant playlists, media playlists, or MPDs to the CDN.
5. The CDN forwards multivariant playlists, media playlists, or MPDs to MediaTailor.
6. MediaTailor requests ads from the ad decision server (ADS).
7. MediaTailor personalizes manifests by inserting ads into multivariant playlists, media playlists, or MPDs and delivers them directly to viewers.
8. This architecture introduces additional latency, potential caching issues, and complicates troubleshooting.



Request and response flow

When implementing dynamic ad insertion with a CDN, configure your system to support this request and response flow:

1. Configure your player to request multivariant playlists (HLS) or MPDs (DASH) from your CDN with MediaTailor as the manifest origin.
2. Set up your CDN to forward all multivariant playlist, media playlist, and MPD requests to MediaTailor, including all query parameters and headers.
3. Ensure MediaTailor can communicate with your ad decision server (ADS), passing along query parameters and headers.
4. Configure your ADS to use query parameters to determine which ads to insert.

5. Set up the CDN prefix on the MediaTailor playback configuration so MediaTailor can substitute CDN domain names for content and ad segment URL prefixes.
6. Configure your CDN to forward personalized multivariant playlists, media playlists, and MPDs from MediaTailor to the requesting player.
7. Set up your CDN to translate segment URLs, forwarding content segment requests to the origin server and ad requests to the Amazon S3 bucket where MediaTailor stores transcoded ads.

CDN terminology for ad insertion

Understanding these key terms will help you implement and troubleshoot your ad insertion CDN integration:

Origin CDN and edge CDN

Origin CDN: A CDN positioned between MediaTailor and your content origin. It caches content segments to reduce load on your origin servers. In a multi-CDN architecture, this is the first CDN layer that interfaces directly with the origin.

Edge CDN: A CDN positioned between viewers and MediaTailor. It delivers personalized manifests and content to viewers. In a multi-CDN architecture, this is the outermost CDN layer that interfaces directly with viewers.

CDN configuration terms

Cache behavior: Rules that determine how a CDN handles different types of requests. These rules include:

- Caching duration settings
- Origin routing configurations
- Request handling parameters

TTL (Time To Live): The duration for which content remains valid in a CDN cache before it needs to be refreshed from the origin.

Cache key: The unique identifier used by a CDN to store and retrieve cached content. It typically includes:

- URL path
- Query parameters
- Selected headers

Origin shield: An intermediate caching layer between CDN edge locations and your origin server. It reduces the number of requests to your origin.

Request collapsing: A CDN feature that combines multiple simultaneous requests for the same content into a single origin request.

MediaTailor-specific CDN terms

CDN content segment prefix: The CDN domain name that AWS Elemental MediaTailor uses when generating URLs for content segments in manifests.

CDN ad segment prefix: The CDN domain name that MediaTailor uses when generating URLs for ad segments in manifests.

For more information about CDN configuration with MediaTailor, see [Set up CDN integration](#).

Note

These terms are consistent with those used in channel assembly documentation. For channel assembly terminology, see [CDN terminology for channel assembly](#).

Set up basic MediaTailor SSAI with a CDN for optimal ad delivery

This section provides step-by-step instructions for configuring AWS Elemental MediaTailor dynamic ad insertion with a content delivery network (CDN) to optimize your video monetization workflow.

For advanced ad server configuration using dynamic variables, see [MediaTailor dynamic ad variables for ADS requests](#). For information about passing parameters through CDNs for ad targeting, see [MediaTailor manifest query parameters](#).

For conceptual information about SSAI with CDNs, see [Understand ad insertion architecture for CDN and MediaTailor integrations](#).

Prerequisites

Before setting up ad insertion with a CDN, ensure you have:

- An active AWS Elemental MediaTailor configuration
- A content origin server delivering HLS or DASH content with appropriate ad markers

For information about ad markers, see [Understanding ad insertion behavior](#).

- An ad decision server (ADS) that supports VAST or VMAP for ad targeting
- A CDN account (such as Amazon CloudFront or another CDN provider)
- Basic knowledge of manifest manipulation and dynamic ad insertion concepts

Step 1: Configure CDN caching for optimal ad delivery

Proper CDN caching configuration is critical for optimal performance of your video monetization workflow. The caching requirements differ between server-side ad insertion (SSAI) and server-guided ad insertion (SGAI). Use these recommended settings to ensure efficient delivery of both content and personalized advertising:

SSAI CDN caching settings

For server-side ad insertion workflows, proper caching configuration is critical for optimal performance. SSAI requires specific TTL values and cache key settings to ensure personalized manifests are not cached while segments are cached efficiently.

For detailed SSAI caching settings including TTL values, path patterns, and cache key configurations, see [Server-side ad insertion \(SSAI\) caching](#) in the CDN optimization guide.

Key caching principles for SSAI:

- **Manifests:** Set TTL to 0 seconds to prevent caching of personalized content
- **Segments:** Cache aggressively (24+ hours) to reduce origin load
- **Cache keys:** Include all query parameters for manifests, URL path only for segments

SGAI CDN caching settings

For server-guided ad insertion workflows, caching requirements differ from SSAI because SGAIs manifests can be cached for short periods while still providing personalized ad experiences.

For comprehensive SGAIs caching settings including VOD and live TTL values, see the optimization guide's caching tables. SGAIs allows for better cache efficiency than SSAI while maintaining ad personalization capabilities.

Key SGAIs caching differences:

- **Manifests:** Can be cached for short periods (5-30 minutes for VOD, 2-10 seconds for live)
- **Segments:** Cache aggressively like SSAI (24+ hours for most content)
- **Performance benefit:** Better cache hit ratios than SSAI due to cacheable manifests

For Amazon CloudFront, you can implement these settings using cache behaviors with different TTL values and cache key policies. For other CDNs, refer to their specific documentation for implementing similar caching rules.

Step 2: Implement hybrid approaches (if needed)

If your architecture requires a hybrid approach with a separate CDN or caching layer between the content origin and MediaTailor:

1. Implement clear separation of concerns in your CDN configuration.
2. Configure specific CDN settings to prevent the technical issues described in the previous section.
3. Thoroughly test your configuration to verify that manifest personalization functions correctly.
4. Monitor performance metrics to ensure optimal delivery of multivariant playlists, media playlists, MPDs, and segments.

When implementing a hybrid approach, consider these specific configurations:

- For the CDN between content origin and MediaTailor:
 - Configure compression passthrough for manifest files to preserve the original compression state from your origin
 - Include all query parameters in the cache key
 - Set short TTL values for live content manifests
- For the CDN between MediaTailor and viewers:
 - Configure longer cache times for ad segments
 - Set appropriate TTLs for personalized manifests
 - Implement proper origin routing for content vs. ad segments

Step 3: Complete your CDN setup

After choosing your architecture and understanding the request flow, complete your setup by following the detailed configuration steps in [Set up CDN integration](#).

For specific CDN providers, refer to these additional resources:

- Amazon CloudFront: See [CloudFront integration](#) for CloudFront-specific configuration steps
- Other CDNs: Apply the general principles outlined in this guide, adapting them to your specific CDN's configuration options

Step 4: Verify your configuration

After completing your CDN setup, verify that your dynamic ad insertion workflow is functioning correctly:

1. Test playback through your CDN with a sample player
2. Verify that personalized advertising is inserted correctly at designated ad break points
3. Check CDN logs to confirm proper request routing
4. Monitor cache hit rates to ensure optimal performance for both content and ad segments
5. Confirm that ad targeting parameters are being properly passed through the workflow

For comprehensive testing and validation procedures, see [Testing and validation for CDN and MediaTailor integrations](#). For detailed information on monitoring your SSAI implementation, see [Monitor operations for CDN and MediaTailor integrations](#). To optimize performance, see [Optimize performance for CDN and MediaTailor integrations](#).

Integrate MediaTailor SSAI with channel assembly for monetized linear channels

This topic explains how to combine AWS Elemental MediaTailor server-side ad insertion with channel assembly and content delivery network (CDN) integration. This integration enables you to:

- Create monetized linear channels with personalized advertising
- Deliver targeted ads to different viewers watching the same content
- Maintain broadcast-quality viewing experiences

Benefits of combining SSAI with channel assembly

Integrating SSAI with channel assembly provides several key benefits:

Monetization of linear channels

Insert personalized ads into your linear channels to generate revenue from your content library. You can monetize both live and VOD content within a single linear stream.

Personalized advertising

Deliver different ads to different viewers watching the same channel content. This targeted approach increases ad relevance and potential revenue compared to traditional broadcast advertising.

Simplified ad break management

Define ad break points in your channel assembly programs without needing to condition the content with SCTE-35 markers. This makes it easier to insert ads at natural break points in your content.

Broadcast-quality experience

MediaTailor maintains a high-quality viewing experience with seamless transitions between content and ads. Server-side ad insertion eliminates many common issues such as:

- Buffering during ad transitions
- Ad blockers preventing monetization
- Inconsistent playback quality

Scalable delivery

When combined with a CDN, this integration can scale to millions of concurrent viewers without degradation in performance or personalization capabilities.

Architecture overview

The architecture for combining SSAI with channel assembly typically involves these components:

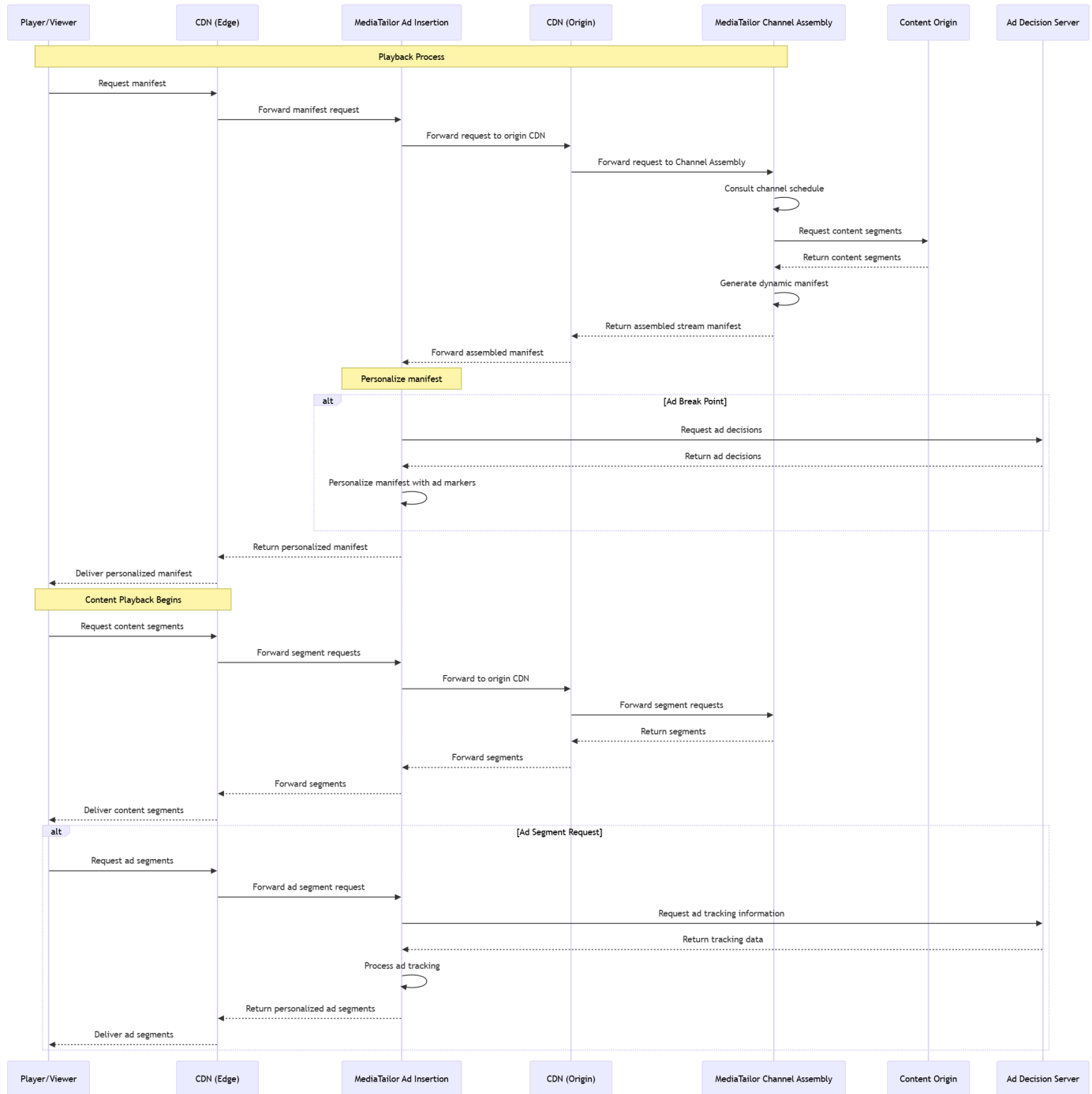
- Channel assembly: Creates linear channels from VOD and live content, and inserts slate content that creates ad markers in the generated manifest
- Ad insertion: Recognizes ad break points and inserts URLs pointing to personalized ad segments in the manifest
- Ad decision server (ADS): Determines which ads to insert for each viewer

- Content delivery network (CDN): Delivers the assembled content and ad segments to viewers
- Origin server: Stores the VOD and live content segments

In this architecture:

1. Channel assembly creates a linear channel from your VOD and live content, and inserts slate content that creates the ad markers in the generated manifest
2. When a viewer requests the channel, ad insertion recognizes the ad breaks that were inserted into the linear channel
3. Ad insertion makes the call to the ADS to receive the list of ads, transcodes them, and inserts URLs pointing to the transcoded ad segments into the personalized manifest
4. The CDN delivers the personalized stream to the viewer

The following diagram illustrates this workflow:



Setting up the integration

To set up SSAI with channel assembly:

1. Configure your edge CDN to accept manifest requests from viewers and forward them to AWS Elemental MediaTailor ad insertion.
2. Set up MediaTailor ad insertion to forward requests to your origin CDN.
3. Configure your origin CDN to forward requests to MediaTailor channel assembly.
4. Set up MediaTailor channel assembly to generate dynamic manifests based on the current schedule.
5. Configure your origin CDN to forward the assembled manifests to MediaTailor ad insertion.
6. Set up MediaTailor ad insertion to request ad decisions from your ad decision server at ad break points.
7. Configure MediaTailor ad insertion to personalize manifests by replacing the ad markers (from channel assembly) with URLs pointing to targeted ad segments (from the ADS).
8. Set up your edge CDN to deliver personalized manifests to viewers.
9. Configure your CDN architecture to handle both content and ad segment requests efficiently.

Defining ad breaks in channel assembly

When creating programs in channel assembly, you can define ad breaks in several ways:

Program transitions

Insert ads between programs in your channel schedule. This is the simplest approach and ensures ads don't interrupt program content.

SCTE-35 markers

If your VOD content contains SCTE-35 markers, channel assembly can preserve these markers, and ad insertion can use them as ad break points.

Time-based insertion

Define ad breaks at specific time points within programs. This allows you to insert ads at natural break points in your content.

For detailed information on creating programs with ad breaks, see [Working with programs](#).

CDN caching considerations

For optimal performance when combining channel assembly and SSAI with a CDN:

- Configure cache behaviors that distinguish between channel assembly and SSAI requests
- Set appropriate TTL values for manifests and segments as recommended in [Step 1: Configure CDN caching for optimal ad delivery](#)
- Ensure proper routing between channel assembly, ad insertion, and your CDN origins
- Monitor the performance metrics for both channel assembly and ad insertion components

Recommended caching settings for combined implementation

Content type	TTL	Cache key elements
Channel assembly manifests	0 seconds	URL path + query parameters
SSAI personalized manifests	0 seconds	URL path + all query parameters
Content segments	24+ hours	URL path only
Ad segments	24+ hours	URL path only

Monitoring the integrated solution

To ensure your integrated solution is performing optimally, monitor these key metrics:

Channel assembly metrics

Monitor manifest generation time, program transitions, and any errors in the channel assembly process.

Ad insertion metrics

Track ad fill rate, ad decision server response times, and ad insertion errors.

CDN metrics

Monitor cache hit rates, origin request volume, and response latency for both content and ad segments.

Viewer experience metrics

Track rebuffering events, startup times, and viewer engagement, especially during ad transitions.

For detailed information on monitoring, see [Monitor operations for CDN and MediaTailor integrations](#) and [Monitor MediaTailor channel assembly CDN operations](#).

Troubleshooting common issues

When troubleshooting issues with the integrated solution, consider these common problems:

Ad break synchronization issues

If ads are not appearing at the expected break points, verify that the ad break definitions in your channel assembly programs are correctly configured and that ad insertion is properly identifying these break points.

Manifest delivery errors

If viewers are experiencing playback issues, check that the CDN is correctly forwarding manifest requests between channel assembly and ad insertion, and that the cache settings are appropriate for the dynamic nature of these manifests.

Segment routing problems

If content or ad segments are not loading, verify that the CDN is correctly routing segment requests to the appropriate origins and that the segment URLs in the manifests are correctly formatted.

Performance degradation

If viewers are experiencing buffering or high latency, check the CDN cache hit rates and origin request volumes to identify potential bottlenecks in the delivery pipeline.

For more troubleshooting guidance, see [Troubleshoot MediaTailor SSAI with CDNs for uninterrupted ad delivery](#).

Best practices

Follow these best practices for a successful integration of SSAI with channel assembly:

- **Test thoroughly:** Test the integrated solution with various content types, ad scenarios, and viewer conditions before deploying to production.
- **Monitor continuously:** Set up comprehensive monitoring and alerting to quickly identify and address any issues that arise.

- **Optimize caching:** Regularly review and adjust your CDN caching settings based on actual usage patterns and performance metrics.
- **Plan for scale:** Design your architecture to handle peak traffic loads, especially for popular channels or events.
- **Consider redundancy:** Implement redundancy in critical components to ensure high availability of your linear channels.
- **Optimize ad transitions:** Ensure smooth transitions between content and ads by using consistent encoding profiles and segment durations.

Related information

For more information about integrating SSAI with channel assembly, see:

Channel assembly documentation

[Using AWS Elemental MediaTailor to create linear assembled streams](#) - Learn about channel assembly concepts

[Channel assembly with CDN](#) - Set up channel assembly with a CDN

SSAI documentation

[Ad insertion with CDN](#) - Set up ad insertion with a CDN

[Understand ad insertion architecture for CDN and MediaTailor integrations](#) - Understand ad insertion CDN architecture

CDN configuration

[Set up CDN integration](#) - General CDN configuration guidance

[CloudFront integration](#) - CloudFront specific configuration

Optimize performance for CDN and MediaTailor integrations

Maximize the performance of your AWS Elemental MediaTailor ad insertion implementation by optimizing your content delivery network (CDN) configuration. These settings ensure efficient content delivery and optimal viewer experience.

For detailed caching and routing optimization guidance specific to SSAI implementations, see [CDN performance optimization](#). The consolidated optimization guide provides comprehensive caching

settings, request routing configuration, and performance benchmarks that apply to all MediaTailor CDN integrations.

Common performance challenges

SSAI implementations with CDNs can face several performance challenges:

Manifest manipulation overhead

MediaTailor performs real-time manifest manipulation, which can introduce latency if not optimized properly. The following can introduce latency:

- Processing time for ad decision server (ADS) requests
- Time required to modify manifests with ad segment references
- Additional processing for personalization

Cache efficiency issues

Personalized manifests can reduce CDN cache efficiency because:

- Each viewer may receive a unique manifest
- Session parameters can fragment the cache
- Dynamic content requires careful cache configuration

Origin load spikes

Improper caching can lead to origin load spikes during:

- High-traffic events
- Cache refreshes
- CDN configuration changes

Ad-related playback issues

Ad insertion can cause playback disruptions such as:

- Buffering during ad transitions
- Quality differences between content and ads
- Playback failures when ads cannot be retrieved

For comprehensive performance optimization guidance including caching strategies, request routing, performance benchmarks, and advanced optimization techniques, see [CDN performance](#)

[optimization](#). The consolidated optimization guide provides detailed settings and benchmarks that apply to all MediaTailor CDN integrations, including SSAI implementations.

Best practices summary

Follow these best practices to ensure optimal SSAI performance with CDNs:

Architecture best practices

- Choose the right architectural pattern for your scale and requirements
- Deploy services in close proximity to minimize latency
- Implement redundancy for critical components

Caching best practices

- Use different caching strategies for different content types
- Optimize cache keys to balance personalization and efficiency
- Set appropriate TTLs based on content type and update frequency. For detailed TTL recommendations, see [Caching optimization for CDN and MediaTailor integrations](#).

Ad delivery best practices

- Optimize ADS interactions with timeouts and fallbacks
- Prepare ads to match content specifications
- Implement efficient ad segment delivery

Monitoring best practices

- Monitor all components of your SSAI implementation
- Set up alerts for performance degradation
- Regularly review and optimize your configuration

Complete optimization guidance

For comprehensive CDN optimization guidance including detailed caching strategies, request routing configuration, performance benchmarks, and advanced optimization techniques, see [CDN performance optimization](#). The consolidated optimization guide provides complete settings and benchmarks that apply to all MediaTailor CDN integrations, including SSAI implementations.

Monitor operations for CDN and MediaTailor integrations

AWS Elemental MediaTailor provides robust analytics capabilities that, when combined with content delivery network (CDN) metrics, offer comprehensive insights into your SSAI implementation. This topic covers:

For comprehensive CDN monitoring guidance including essential metrics, monitoring tools setup, alert configuration, and troubleshooting strategies that apply to all MediaTailor implementations, see [CDN monitoring](#). This topic focuses on SSAI-specific monitoring requirements and ad insertion analytics.

- Monitoring strategies for SSAI and CDN integration
- Analytics tools and data collection methods
- Data-driven optimization techniques

Key metrics for SSAI with CDNs

To effectively monitor your SSAI implementation with CDNs, track these essential metrics:

Ad insertion metrics

Ad fill rate: The percentage of ad opportunities that were successfully filled with ads.

Ad error rate: The percentage of ad requests that resulted in errors.

Ad response time: How long it takes for the ad decision server to respond to ad requests.

Ad duration accuracy: How closely the actual duration of inserted ads matches the expected duration.

Viewer experience metrics

Rebuffering ratio: The percentage of viewing time spent buffering.

Start-up time: How long it takes for video playback to begin.

Ad transition smoothness: How seamlessly the player transitions between content and ads.

Session duration: How long viewers watch before abandoning the stream.

Analytics tools and integration

Combine these tools to create a comprehensive analytics solution for your SSAI implementation:

AWS Elemental MediaTailor server-side metrics

MediaTailor provides built-in metrics through Amazon CloudWatch that track ad requests, responses, and errors. These metrics can be viewed in the CloudWatch console or integrated into custom dashboards.

Key MediaTailor metrics include:

- `AdDecisionServer.Ads`: The number of ads returned by the ad decision server.
- `AdDecisionServer.Duration`: The total duration of ads returned by the ad decision server.
- `AdDecisionServer.Errors`: The number of errors returned by the ad decision server.
- `AdDecisionServer.Latency`: The response time of the ad decision server.

For a complete list of MediaTailor metrics, see [Monitoring MediaTailor with Amazon CloudWatch](#).

CDN analytics

CDN providers offer detailed analytics on content delivery performance. For Amazon CloudFront, use CloudWatch metrics and Amazon CloudFront access logs to analyze delivery patterns.

Important CDN metrics to monitor:

- Request count by content type (manifests vs. segments)
- Cache hit ratio for different content types
- Geographic distribution of viewers
- Error rates by error code

Client-side tracking

Implement client-side tracking to gather viewer experience metrics that aren't available server-side:

- Player events (play, pause, seek, buffer)

- Ad view completion rates
- Quality of service metrics (resolution changes, bitrate)
- Viewer engagement patterns

Consider using MediaTailor client-side tracking to collect and report these metrics.

Integrated dashboards

Create comprehensive dashboards that combine metrics from multiple sources:

- Use CloudWatch dashboards to combine MediaTailor and CloudFront metrics
- Consider third-party analytics platforms for more advanced visualization
- Set up cross-service correlation to identify relationships between metrics

Implementing a monitoring strategy

Follow these steps to implement a comprehensive monitoring strategy for your SSAI with CDN implementation:

1. Set up basic monitoring

- Enable CloudWatch metrics for MediaTailor
- Configure CDN logging and metrics collection
- Implement client-side tracking in your video player

2. Create custom dashboards

- Build a CloudWatch dashboard that combines key metrics
- Include visualizations for ad fill rate, CDN performance, and viewer experience
- Add annotations for important events (configuration changes, major broadcasts)

3. Configure alerts

- Set up CloudWatch alarms for critical metrics
- Create composite alarms that trigger on multiple related conditions
- Configure notification channels (email, SMS, Amazon SNS)

4. Implement automated responses

- Use CloudWatch Events to trigger automated responses to common issues
- Create runbooks for manual intervention when needed
- Document troubleshooting procedures for different alert scenarios

Example Creating a comprehensive SSAI monitoring dashboard

This example shows how to create a CloudWatch dashboard that combines MediaTailor and CloudFront metrics:

```
{
  "widgets": [
    {
      "type": "metric",
      "properties": {
        "metrics": [
          [ "AWS/MediaTailor", "AdDecisionServer.Ads", "Configuration", "your-config-
name" ],
          [ ".", "AdDecisionServer.Errors", ".", "." ]
        ],
        "period": 300,
        "stat": "Sum",
        "region": "us-west-2",
        "title": "Ad Decision Server Performance"
      }
    },
    {
      "type": "metric",
      "properties": {
        "metrics": [
          [ "AWS/CloudFront", "Requests", "DistributionId", "your-distribution-id" ],
          [ ".", "4xxErrorRate", ".", "." ],
          [ ".", "5xxErrorRate", ".", "." ]
        ],
        "period": 300,
        "stat": "Average",
        "region": "us-east-1",
        "title": "CDN Performance"
      }
    }
  ]
}
```

Data-driven optimization

Use the analytics data you collect to optimize your SSAI implementation:

CDN cache optimization

Analyze cache hit ratios to identify opportunities for improvement:

- Adjust TTL settings based on content type and update frequency. For detailed TTL recommendations, see [Step 1: Configure CDN caching for optimal ad delivery](#).
- Optimize cache key settings to improve cache efficiency
- Consider implementing origin shield for multi-layered caching

Ad delivery optimization

Use ad performance metrics to improve ad delivery:

- Identify and address common ad insertion errors
- Optimize ad decision server response times
- Adjust ad targeting parameters based on fill rate analysis

Viewer experience optimization

Improve the viewer experience based on client-side metrics:

- Analyze drop-off patterns during ad breaks
- Optimize ad transition points for smoother playback
- Adjust ad frequency and duration based on viewer engagement data

Cost optimization

Balance performance and cost considerations:

- Analyze bandwidth usage patterns to optimize CDN costs
- Consider price class adjustments for CloudFront distributions
- Evaluate the cost-benefit of different caching strategies

Best practices

Follow these best practices for effective SSAI monitoring and analytics:

- **Establish baselines:** Collect metrics during normal operation to establish performance baselines that can be used for comparison during troubleshooting.
- **Implement multi-level monitoring:** Monitor at different levels of your architecture (origin, CDN, player) to get a complete picture of performance.

- **Correlate metrics across services:** Look for relationships between metrics from different services to identify root causes of issues.
- **Use anomaly detection:** Implement CloudWatch anomaly detection to automatically identify unusual patterns in your metrics.
- **Regularly review and refine:** Schedule regular reviews of your monitoring strategy and adjust based on changing requirements and new insights.
- **Document findings and actions:** Maintain a record of optimization efforts and their results to build institutional knowledge.

Related information

For more information about monitoring and analytics for SSAI with CDNs, see:

- [Optimize performance for CDN and MediaTailor integrations](#) for performance optimization techniques
- [Troubleshoot MediaTailor SSAI with CDNs for uninterrupted ad delivery](#) for troubleshooting common issues
- [Monitoring MediaTailor with Amazon CloudWatch](#) for detailed information about MediaTailor metrics
- [Viewing CloudFront and edge function metrics](#) for information about CloudFront metrics

Troubleshoot MediaTailor SSAI with CDNs for uninterrupted ad delivery

This section provides solutions for common issues when using AWS Elemental MediaTailor dynamic ad insertion with a content delivery network (CDN). These solutions will help you troubleshoot problems with your video monetization through personalized advertising.

For comprehensive CDN troubleshooting guidance including cache performance issues, HTTP error resolution, testing procedures, and diagnostic techniques that apply to all MediaTailor implementations, see [Troubleshoot CDN integration](#). This section focuses on SSAI-specific troubleshooting requirements and ad insertion issues.

If you encounter issues with your CDN and SSAI setup, check these common problems:

Personalized advertising not appearing in stream

Verify that your ADS responds correctly and that AWS Elemental MediaTailor communicates with it. Check these potential issues:

- Ad targeting query parameters not properly forwarded through your CDN
- Ad break points incorrectly defined in your content
- ADS connectivity or response problems

Playback errors at ad break points

Ensure that ad segments are properly transcoded to match your content bitrates and resolutions. Check these common issues:

- CDN incorrectly routing requests for ad segments
- Manifest manipulation errors at transition points
- Mismatched encoding profiles between content and ads

Stale manifests

For live content, verify that your CDN cache TTL settings are appropriate. For personalized manifests, use TTL of 0 seconds. Consider implementing cache invalidation for rapidly changing manifests. For comprehensive TTL guidance, see [Caching optimization for CDN and MediaTailor integrations](#).

High latency

Check your CDN configuration for optimal routing. Ensure that your CDN has edge locations near your viewers for best performance.

Advanced troubleshooting

For more complex issues, try these advanced troubleshooting techniques:

Inconsistent ad targeting behavior

Check for query parameter inconsistencies between player requests and ADS requests. Ensure all required targeting parameters are properly passed through the CDN.

CDN cache inconsistencies

Verify cache key configurations to ensure proper content differentiation. Consider implementing cache purging for critical manifest updates.

Ad tracking failures

Check that beacon URLs are properly forwarded and not blocked by the CDN. Verify that client players can reach the tracking endpoints.

Performance optimization

To optimize the performance of your dynamic ad insertion and video monetization workflow:

- Fine-tune TTL settings based on content type and viewer patterns. For detailed TTL recommendations, see [Caching optimization for CDN and MediaTailor integrations](#).
- Implement geographic routing to minimize latency for global audiences
- Consider using multiple CDNs for redundancy and optimal performance
- Monitor cache hit ratios and adjust configurations accordingly
- Optimize manifest manipulation processes to reduce processing time at ad break points
- Pre-transcode ads to match common content profiles for seamless personalized advertising insertion

Performance benchmarks

When optimizing your AWS Elemental MediaTailor ad insertion CDN integration, aim for these performance benchmarks:

Cache Hit Ratio Targets

Content segments: greater than 95% cache hit ratio

Ad segments: greater than 90% cache hit ratio

Manifests: Not applicable (should not be cached for personalized ad insertion)

Latency Benchmarks

Manifest request latency: less than 100ms (P95)

Content segment delivery: less than 50ms (P95)

Ad segment delivery: less than 75ms (P95)

End-to-end startup time: less than 2 seconds

Origin Load Metrics

Origin requests per viewer: less than 0.1 requests per minute per viewer

Origin bandwidth per viewer: less than 5% of total viewer bandwidth

Error Rate Targets

Manifest errors: less than 0.1%

Segment errors: less than 0.01%

Player-reported rebuffering: less than 1%

Scalability Benchmarks

Support for 10 times the normal traffic during peak events without degradation

Ability to handle greater than 1000 requests per second per channel

Use Amazon CloudWatch metrics to track these performance indicators. For detailed monitoring instructions, see [Monitoring AWS Elemental MediaTailor with Amazon CloudWatch metrics](#).

Related information

For more information about ad insertion with CDNs, see:

Ad insertion documentation

[Getting started with MediaTailor ad insertion](#) - Learn about ad insertion concepts

[Setting up](#) - Get started with ad insertion

CDN integration

[Set up CDN integration](#) - General CDN configuration guidance

[CloudFront integration](#) - CloudFront specific configuration

Channel assembly integration

[Channel assembly with CDN](#) - Learn about channel assembly with CDNs

[Implement ad insertion](#) - Implement ad insertion with channel assembly

Monitoring and optimization

[Monitor operations for CDN and MediaTailor integrations](#) - Comprehensive monitoring and analytics

[Optimize performance for CDN and MediaTailor integrations](#) - Performance optimization guide

[Monitoring AWS Elemental MediaTailor with Amazon CloudWatch metrics](#) - CloudWatch metrics for MediaTailor

Build MediaTailor linear channels with channel assembly and CDN

This section provides comprehensive guidance for integrating AWS Elemental MediaTailor channel assembly with a content delivery network (CDN). Follow these steps to set up, configure, and optimize your channel assembly CDN integration.

You can also combine channel assembly with server-side ad insertion (SSAI) to create monetized linear channels with personalized advertising. This powerful integration enables you to deliver targeted ads to different viewers watching the same channel content, increasing your revenue opportunities while maintaining a broadcast-quality viewing experience. For information about SSAI with CDNs, see [Ad insertion with CDN](#).

In this topic, we use the term *manifests* to refer collectively to multivariant playlists, media playlists, and MPDs.

For more information about MediaTailor channel assembly, see [Using AWS Elemental MediaTailor to create linear assembled streams](#).

What you'll need

Before setting up MediaTailor channel assembly with a CDN, gather these required resources:

AWS account and permissions

An AWS account with appropriate permissions to create and manage MediaTailor resources

IAM permissions for MediaTailor, CloudFront (if using), and related services

For detailed permission requirements, see [Security in AWS Elemental MediaTailor](#).

Required services

A running MediaTailor channel assembly channel (not just an SSAI configuration)

A content delivery network (CDN) account (Amazon CloudFront or third-party CDN)

Origin storage for your VOD content (Amazon S3, MediaPackage, or other origin server)

Content requirements

VOD sources properly encoded and packaged in HLS or DASH format. For information about working with source locations and VOD sources, see [Working with source locations](#).

Content with consistent segment durations (recommended minimum: 1 second)

Ad slate content for ad breaks (if implementing ad insertion). For information about configuring slate, see [MediaTailor slate ad insertion](#).

Before you begin

Important

This workflow requires a running MediaTailor channel assembly channel. Having only a MediaTailor SSAI configuration is not sufficient for this integration. You must have an active channel assembly channel configured and operational before proceeding with CDN integration.

Complete these setup tasks before implementing MediaTailor channel assembly with a CDN:

1. Configure network connectivity between your CDN, MediaTailor, and origin servers
2. Set up HTTPS for secure content delivery
3. Configure DNS settings for your CDN domain

Knowledge prerequisites

To successfully implement this solution, you should have:

- Understanding of streaming protocols (HLS/DASH)
- Basic knowledge of CDN configuration principles
- Familiarity with MediaTailor channel assembly concepts

For basic MediaTailor setup, refer to [Setting up](#) and [Getting started with MediaTailor channel assembly](#). For information about working with source locations and VOD sources, see [Working with source locations](#). For information about configuring slate, see [MediaTailor slate ad insertion](#).

Benefits of CDN integration

Integrating channel assembly with a CDN delivers these key benefits.

Improved viewer experience

Properly configured CDNs reduce buffering, startup times, and playback errors for linear channels. This results in higher viewer engagement and satisfaction.

Cost reduction

Efficient caching strategies minimize origin requests. This reduces data transfer costs and origin server load, particularly important for high-volume linear channels.

Scalability

Optimized CDN configurations handle traffic spikes during popular events without degrading performance, ensuring your linear channels remain available even during peak viewing times.

Global reach

Properly configured CDNs deliver content with low latency to viewers worldwide, regardless of their location, expanding your potential audience.

Seamless program transitions

Optimized CDN configuration ensures smooth transitions between programs in your linear channel, creating a broadcast-quality viewing experience.

Topics

- [Understand MediaTailor channel assembly CDN architecture](#)
- [Set up basic MediaTailor channel assembly with a CDN](#)

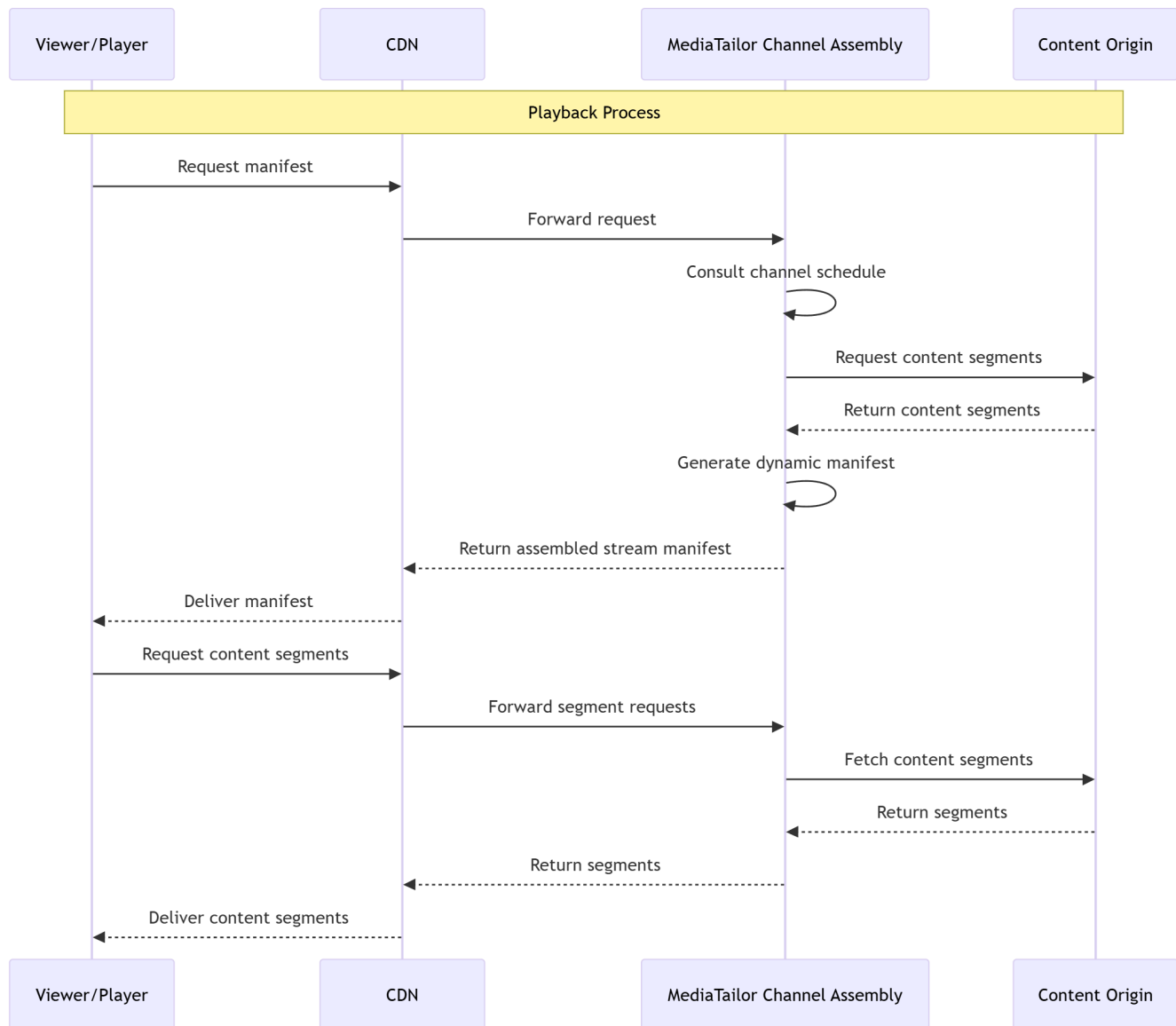
- [Configure base URLs for MediaTailor channel assembly CDN](#)
- [Implement MediaTailor ad insertion with channel assembly](#)
- [Configure time-shifted viewing for MediaTailor channel assembly](#)
- [Monitor MediaTailor channel assembly CDN operations](#)
- [Complete channel assembly CDN optimization](#)

Understand MediaTailor channel assembly CDN architecture

AWS Elemental MediaTailor channel assembly integrates with content delivery networks (CDNs) to deliver linear streaming channels with improved performance and global reach. The recommended architecture positions the CDN between viewers and channel assembly, with channel assembly accessing content directly from your origin. This topic explains the core architecture components and how they work together to deliver your content.

1. Viewers request manifests from the CDN
2. CDN forwards requests to channel assembly
3. Channel assembly assembles the manifests from VOD sources
4. Channel assembly returns the manifests to the CDN, which forwards them to viewers
5. Viewers request segments through the CDN
6. CDN routes segment requests to the appropriate origin

This architecture ensures optimal performance while maintaining the security and flexibility benefits of using a CDN.



CDN terminology for channel assembly

Understanding these key terms will help you implement and troubleshoot your channel assembly CDN integration:

Origin CDN and edge CDN

Origin CDN: A CDN positioned between MediaTailor and your content origin. It caches content segments to reduce load on your origin servers. In a multi-CDN architecture, this is the first CDN layer that interfaces directly with the origin.

Edge CDN: A CDN positioned between viewers and MediaTailor. It delivers personalized manifests and content to viewers. In a multi-CDN architecture, this is the outermost CDN layer that interfaces directly with viewers.

CDN configuration terms

Cache behavior: Rules that determine how a CDN handles different types of requests, including caching duration and origin routing.

TTL (Time To Live): The duration for which content remains valid in a CDN cache before it needs to be refreshed from the origin. For detailed TTL recommendations, see [Caching optimization for CDN and MediaTailor integrations](#).

Cache key: The unique identifier used by a CDN to store and retrieve cached content, often including URL path, query parameters, and headers.

Origin shield: An intermediate caching layer between CDN edge locations and your origin server that reduces the number of requests to your origin.

Request collapsing: A CDN feature that combines multiple simultaneous requests for the same content into a single origin request.

MediaTailor-specific CDN terms

CDN content segment prefix: The CDN domain name that MediaTailor uses when generating URLs for content segments in manifests.

CDN ad segment prefix: The CDN domain name that MediaTailor uses when generating URLs for ad segments in manifests.

For more information about CDN configuration with MediaTailor, see [Set up CDN integration](#).

Set up basic MediaTailor channel assembly with a CDN

AWS Elemental MediaTailor channel assembly enables you to configure a basic integration with your content delivery network (CDN) for efficient delivery of linear streaming channels to your viewers. Follow these steps to set up the integration between channel assembly and your CDN.

1. Configure your CDN to accept manifests from viewers and forward them to MediaTailor channel assembly.

2. Set up MediaTailor channel assembly to access your channel schedule and determine the current programming.
3. Configure MediaTailor channel assembly to request content segments from your origin server based on the schedule.
4. Ensure your content origin can deliver the requested segments to MediaTailor channel assembly.
5. Set up MediaTailor channel assembly to generate dynamic manifests based on the current schedule.
6. Configure your CDN to deliver the assembled multivariant playlists, media playlists, and MPDs to viewers.
7. Set up your CDN to handle segment requests from viewers, with appropriate cache settings.
8. Configure your CDN to forward cache misses to MediaTailor channel assembly.
9. Set up MediaTailor channel assembly to retrieve requested segments from your content origin.
10. Configure your CDN to deliver content segments to viewers for playback.

Configure base URLs for MediaTailor channel assembly CDN

AWS Elemental MediaTailor channel assembly requires proper base URL configuration to ensure content routing through your content delivery network (CDN) works correctly. Configure the base URL settings in channel assembly to enable successful content delivery to viewers.

Content segment URL configuration

In your channel assembly channel configuration, set the **Base URL** to your CDN domain. This ensures that all segment URLs in the assembled manifest point to your CDN rather than directly to your origin server.

For example, if your origin content is at `http://origin.example.com/content/` and your CDN domain is `https://cdn.example.com/`, set the Base URL to `https://cdn.example.com/content/`.

Access restriction configuration

To enhance security, configure your CDN to restrict direct access to your origin server:

1. Set up origin access controls in your CDN.
2. Configure your origin server to only accept requests from your CDN.

3. Use signed URLs or cookies for viewer authentication if needed.

For Amazon CloudFront, you can use Origin Access Control (OAC) to secure access to your origin. For more information about securing your CloudFront integration, see [CloudFront integration](#).

Implement MediaTailor ad insertion with channel assembly

Channel assembly in AWS Elemental MediaTailor integrates seamlessly with server-side ad insertion (SSAI) and content delivery networks (CDNs) to create monetized linear channels with personalized advertising.

When you combine channel assembly with SSAI, you can build linear channels that deliver personalized ads to viewers while maintaining broadcast-quality experiences. This integration enables you to do the following:

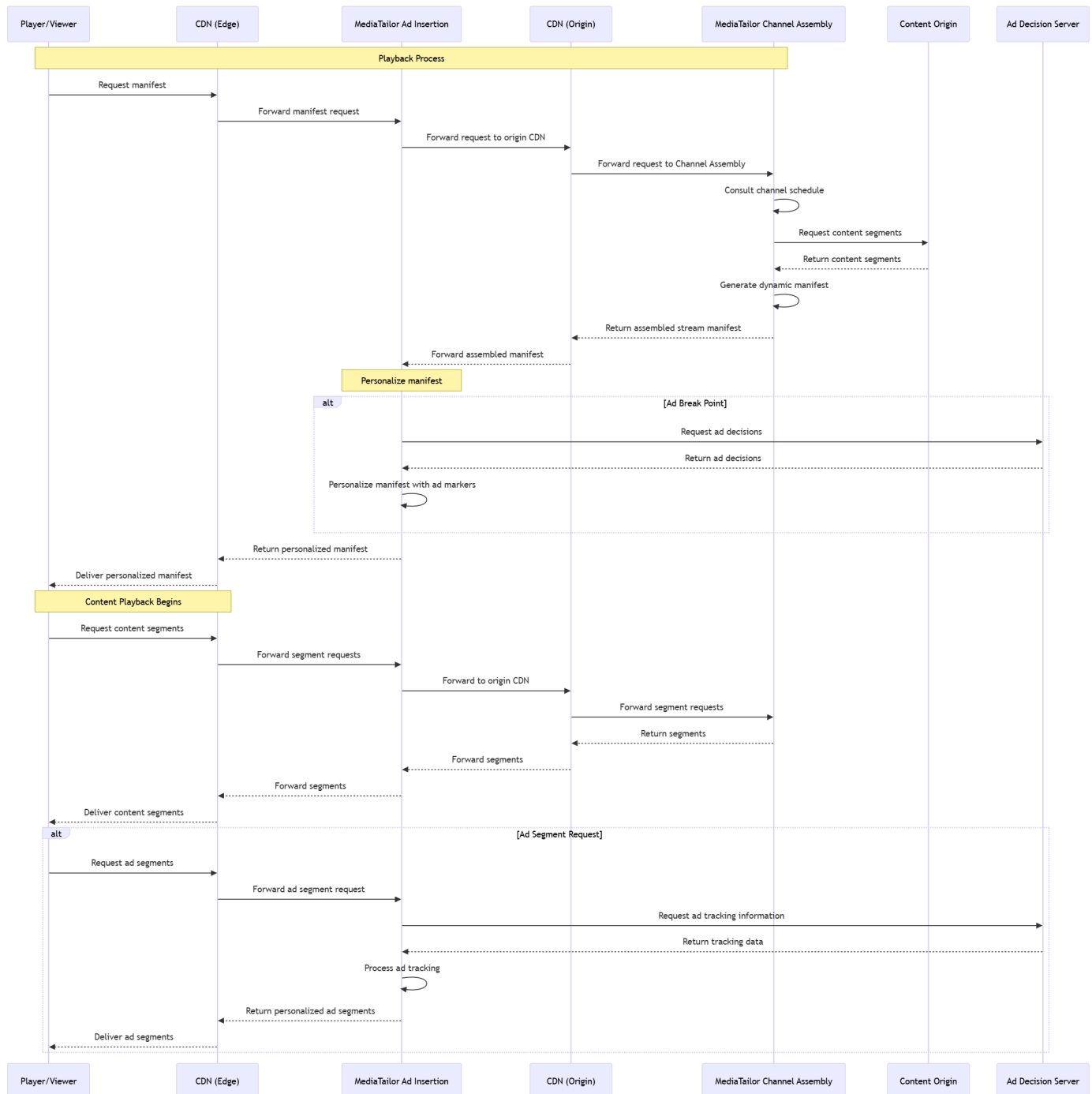
- Monetize content - Generate revenue through targeted advertising in your linear channels
- Personalize experiences - Deliver different ads to viewers watching the same channel based on their profiles
- Maintain quality - Ensure seamless transitions between content and ads for broadcast-quality viewing
- Scale efficiently - Support millions of concurrent viewers through CDN delivery

For detailed information about SSAI with CDNs, see [Ad insertion with CDN](#).

1. Configure your edge CDN to accept manifest requests from viewers and forward them to MediaTailor ad insertion.
2. Set up MediaTailor ad insertion to forward requests to your origin CDN.
3. Configure your origin CDN to forward requests to MediaTailor channel assembly.
4. Set up MediaTailor channel assembly to generate dynamic manifests based on the current schedule.
5. Configure your origin CDN to forward the assembled manifests to MediaTailor ad insertion.
6. Set up MediaTailor ad insertion to request ad decisions from your ad decision server at ad break points.
7. Configure MediaTailor ad insertion to personalize manifests with ad markers.
8. Set up your edge CDN to deliver personalized manifests to viewers.

9. Configure your CDN architecture to handle both content and ad segment requests efficiently.

The following diagram illustrates this combined workflow:



For optimal performance when combining channel assembly and SSAI:

- Configure cache behaviors that distinguish between channel assembly and SSAI requests
- Set appropriate TTL values for manifests and segments as recommended in [Caching optimization for CDN and MediaTailor integrations](#)
- Ensure proper routing between channel assembly, ad insertion, and your CDN origins
- Monitor the performance metrics for both channel assembly and ad insertion components

For detailed information about configuring SSAI with CDNs, see:

- [Understand ad insertion architecture for CDN and MediaTailor integrations](#) - Learn about SSAI architecture and concepts
- [Set up basic MediaTailor SSAI with a CDN for optimal ad delivery](#) - Step-by-step SSAI configuration instructions
- [Troubleshoot MediaTailor SSAI with CDNs for uninterrupted ad delivery](#) - Troubleshoot common SSAI integration issues

Configure time-shifted viewing for MediaTailor channel assembly

AWS Elemental MediaTailor channel assembly supports time-shifted viewing capabilities that enable DVR-like functionality such as pause, rewind, and start-over for your linear channels. Enable these features by configuring your content delivery network (CDN) to support time-shifted viewing, which allows viewers to control their viewing experience.

Understanding time-shifted viewing

Time-shifted viewing enables DVR-like functionality for linear channels, including:

- **Start-over:** Viewers can start watching a program from the beginning, even if they join after it has started
- **Pause and resume:** Viewers can pause content and resume watching later
- **Rewind and fast-forward:** Viewers can navigate backward and forward through available content
- **Delayed viewing:** Viewers can watch content that aired earlier in the channel's schedule

Time-shifted viewing works by adding a `start` parameter to the channel's playback URL. The parameter specifies an offset in seconds relative to the current time:

- Negative values indicate a time in the past (such as `start=-3600` means "start from 1 hour ago")
- Positive values indicate a time in the future (such as `start=3600` means "start from 1 hour in the future")

Example URL with time-shift parameter:

```
https://example-cdn.com/out/v1/channel-name/index.m3u8?start=-3600
```

Time delay resolution

To support time-shifted viewing with a CDN:

1. Configure your CDN to forward the `start` query parameter to channel assembly.
2. Set up cache behaviors that include the `start` parameter in the cache key.
3. For manifests with time-shift parameters, use a short TTL or no caching.

This ensures that each viewer receives the correct manifest for their requested time position.

CDN requirements for time-shifting

Your CDN must meet these requirements to support time-shifted viewing with channel assembly:

- Forward all query parameters to channel assembly.
- Include the `start` parameter in the cache key.
- Support proper cache invalidation for time-shifted manifests.
- Handle varying manifest responses based on query parameters.

Monitor MediaTailor channel assembly CDN operations

AWS Elemental MediaTailor channel assembly requires effective monitoring when integrated with a content delivery network (CDN) to ensure reliable content delivery. Implement monitoring strategies for your channel assembly and CDN integration to help ensure reliable content delivery and quick problem resolution.

For comprehensive CDN monitoring guidance including essential metrics, monitoring tools setup, alert configuration, and troubleshooting strategies that apply to all MediaTailor implementations, see [CDN monitoring](#). This section focuses on channel assembly-specific monitoring requirements.

Implement specific monitoring for your channel assembly and CDN integration:

- Track manifest generation metrics in channel assembly.
- Monitor time-shifted viewing requests and their impact on CDN cache hit rates.
- Configure alerts for unusual patterns in manifest requests.
- Implement tracking for segment availability across your content sources.

For Amazon CloudFront, create a dashboard that integrates CDN metrics with MediaTailor metrics to visualize your entire delivery pipeline using [CloudWatch dashboards](#).

If you're also using SSAI with your channel assembly, see [Monitor CDN operations](#) for additional monitoring recommendations specific to ad insertion.

Complete channel assembly CDN optimization

For detailed CDN optimization guidance specific to channel assembly implementations, including caching strategies, request routing configuration, performance benchmarks, and advanced optimization techniques, see [CDN performance optimization](#).

The consolidated optimization guide provides:

- Channel assembly-specific caching settings with recommended TTL values
- Request routing optimization for manifest and segment delivery
- Performance benchmarks and targets for all MediaTailor implementations
- Advanced optimization techniques including Origin Shield and compression
- Combined workflow guidance for channel assembly with SSAI

Integrate MediaTailor with MediaPackage and CDN

AWS Elemental MediaTailor integrates with AWS Elemental MediaPackage to deliver personalized video ads through a content delivery network (CDN). MediaPackage is a just-in-time video packaging and origination service that prepares and protects your video content for delivery over

the internet. It takes your live or on-demand video content and packages it into streaming formats like HLS and DASH, making it ready for viewers on various devices.

When you combine MediaPackage with MediaTailor and a CDN, you create a complete streaming workflow that delivers personalized ads at scale. The CDN distributes your content globally, reducing latency and improving viewer experience, while MediaTailor inserts targeted advertisements into your streams.

This topic focuses on the essential integration steps to get MediaTailor, MediaPackage, and your CDN working together. For advanced configuration options, troubleshooting, and monitoring guidance, see [Next steps](#).

Understanding the MediaPackage and CDN workflow

Before configuring your integration, it's important to understand how MediaPackage, MediaTailor, and your CDN work together:

1. **Content preparation:** MediaPackage receives your live or on-demand video content and packages it into streaming formats (HLS or DASH manifests and segments).
2. **Ad insertion:** MediaTailor requests manifests from MediaPackage, inserts personalized ads, and serves the modified manifests to viewers.
3. **Global distribution:** Your CDN caches and distributes both the content segments (from MediaPackage) and ad segments (from MediaTailor) to viewers worldwide.
4. **Viewer playback:** Video players request manifests through the CDN, which routes requests appropriately between MediaTailor (for manifests) and MediaPackage (for content segments).

This architecture provides several benefits:

- **Scalability:** The CDN handles high viewer loads without impacting your origin servers
- **Performance:** Content is delivered from edge locations closest to viewers
- **Cost efficiency:** Reduced bandwidth costs through caching
- **Reliability:** Multiple edge locations provide redundancy

Prerequisites

Before you begin, ensure you have the following components configured:

1. **MediaPackage endpoint:** A configured MediaPackage endpoint that's receiving and packaging your video content. For setup instructions, see [Getting started with MediaPackage](#) in the MediaPackage user guide.
2. **MediaTailor configuration:** A MediaTailor configuration that uses your MediaPackage endpoint as the content origin. For setup instructions, see [Integrating a content source for MediaTailor ad insertion](#).
3. **CDN distribution:** A CDN distribution (such as CloudFront) configured to work with streaming media. For setup instructions, see [Creating a distribution](#) in the CloudFront developer guide.
4. **Ad decision server:** A configured ad decision server that returns VAST or VMAP responses for ad insertion.

Step 1: Configure essential CDN settings

Proper CDN configuration is critical for successful MediaPackage integration. Incorrect settings can cause playback failures, poor cache performance, and increased costs. Without the right cache policies and query parameter forwarding, your CDN might not deliver manifests correctly or might bypass caching entirely, leading to high origin server load and degraded viewer experience.

Configure basic cache settings

Configuring basic caching is essential because MediaPackage uses specific cache-control headers to optimize content delivery. Without proper cache settings, your CDN might ignore these headers, leading to unnecessary origin requests and increased latency. Follow these steps to ensure optimal caching behavior:

To configure basic caching that works with MediaPackage:

1. Open your CloudFront distribution settings in the CloudFront console.
2. Select or create a cache policy for your MediaPackage origin.
3. Enable the "Origin Cache-Control Headers" option.
4. Allow MediaPackage to control caching behavior through its cache-control headers.

This basic configuration allows MediaPackage to set appropriate cache durations for different content types automatically. To implement advanced cache optimization with specific TTL values and performance tuning, complete this basic setup first, then continue to [Optimize CDN caching for MediaTailor and MediaPackage content delivery](#).

Configure essential query parameters

Query parameter configuration is crucial for MediaPackage functionality. Your CDN must forward specific query parameters to enable features like time-shifted viewing and low-latency streaming. Incorrect query parameter settings can prevent these features from working and I'm concerned reduce cache efficiency. Follow these steps to configure query parameter forwarding:

To ensure your CDN forwards the required query parameters to MediaPackage:

1. In your CloudFront distribution settings, select or create a cache policy for manifest requests.
2. Under "Cache key settings," select "Include specified query strings."
3. Add the following essential query parameters:
 - `start` and `end` - For time-shifted viewing functionality. These parameters are passed through to MediaPackage to define specific content windows for startover and catch-up viewing
 - `_HLS_msn` and `_HLS_part` - For supporting LL-HLS playback requests
 - `m` - For capturing the modified time of the endpoint. MediaPackage responses always include the `?m=###` tag to capture the modified time of the endpoint. If content is already cached with a different value for this tag, CloudFront requests a new manifest instead of serving the cached version
 - `aws.manifestfilter` - For manifest filtering functionality. If you're using manifest filtering, this parameter must be included to configure the distribution to forward the `aws.manifestfilter` query string to the MediaPackage origin, which is required for the manifest filtering feature to work
4. Only include the query strings that MediaPackage uses. Including unnecessary query strings reduces cache efficiency by creating multiple cache variations for the same content.

These parameters enable basic MediaPackage functionality with your CDN. If you need to implement content filtering for different subscription tiers or device types, complete this basic query parameter setup first, then proceed to [Set up manifest filtering with MediaTailor, MediaPackage, and CDN](#).

For information about how MediaTailor passes query parameters like `start` and `end` through to MediaPackage for time-shifted viewing, see [MediaTailor query parameter handling for origins in MediaTailor manifest query parameters](#).

Configure response timeout for LL-HLS

Timeout configuration is critical for low-latency HLS because LL-HLS uses a "blocking requests" mechanism where the CDN waits for new content segments. If your timeout is too short, requests will fail before MediaPackage can respond with new segments, causing playback interruptions and poor viewer experience. Configure appropriate timeouts to ensure smooth LL-HLS playback:

If you're using low-latency HLS, configure your CDN timeout settings:

1. In your CDN settings, locate the origin timeout configuration.
2. Set the response timeout value to at least three times your parts duration.
3. For example, if your parts duration is 0.3 seconds, set the timeout to at least 0.9 seconds.

This ensures the CDN waits long enough for MediaPackage to respond when using the Blocking Requests mechanism.

Step 2: Verify your integration

Testing your integration is essential to ensure that all components work together correctly before your viewers experience issues. A failed integration can result in broken playback, missing ads, or poor performance. This verification process helps you identify and resolve issues in a controlled environment.

After configuring your CDN settings, verify that your integration is working correctly by testing the complete workflow from content request to ad insertion.

Step 2.1: Test basic playback

Basic playback testing verifies that your CDN correctly handles manifest requests and forwards them to MediaTailor. This test helps identify configuration issues with cache policies, query parameter forwarding, and manifest handling. Follow these steps to test basic manifest delivery:

Test that your basic integration is working by requesting a manifest through your CDN:

1. Use a web browser or curl to request a manifest URL through your CDN.
2. Verify that the manifest loads successfully and contains both content and ad segments.
3. Check that content segment URLs in the manifest point to your CDN domain.
4. Confirm that ad segment URLs also point to your CDN domain.

If the manifest loads correctly and contains the expected URLs, your basic integration is working. For comprehensive testing methodologies and advanced validation procedures, see [Testing and validation for CDN and MediaTailor integrations](#). To set up comprehensive monitoring of your integration's performance and health, see [Monitor performance for MediaPackage, CDN, and MediaTailor integrations](#).

Step 2.2: Test video playback

Video playback testing ensures that your complete integration works end-to-end, including ad insertion and content delivery through your CDN. This test verifies that both content segments and ad segments are properly cached and delivered, and that the viewer experience meets your quality standards. Follow these steps to test complete playback functionality:

Test that video playback works correctly with ads inserted:

1. Use a video player (such as Video.js or HLS.js) to play your content through the CDN.
2. Verify that the video plays smoothly without buffering issues.
3. Confirm that ads are inserted at the expected times during playback.
4. Check that both content and ad segments load from your CDN (not directly from origins).

If playback works smoothly with ads, your integration is functioning correctly. For comprehensive testing methodologies and advanced validation procedures, see [Testing and validation for CDN and MediaTailor integrations](#). If you experience any playback issues, buffering, or ad insertion problems, see [Troubleshoot MediaPackage, CDN, and MediaTailor integrations](#).

Next steps

After completing the basic integration, you can implement advanced features and optimizations:

Advanced CDN optimization

For detailed cache optimization, TTL configuration, and performance tuning, see [Optimize CDN caching for MediaTailor and MediaPackage content delivery](#).

Manifest filtering

To implement content filtering for tiered services, device optimization, or access control, see [Set up manifest filtering with MediaTailor, MediaPackage, and CDN](#).

Troubleshooting

If you encounter issues with your integration, see [Troubleshoot MediaPackage, CDN, and MediaTailor integrations](#).

Performance monitoring

To set up comprehensive monitoring and understand key performance metrics, see [Monitor performance for MediaPackage, CDN, and MediaTailor integrations](#).

Set up manifest filtering with MediaTailor, MediaPackage, and CDN

AWS Elemental MediaTailor uses manifest filtering with AWS Elemental MediaPackage to customize which audio and video streams are included in manifests delivered to different viewers through a content delivery network (CDN). This is particularly useful for implementing tiered service offerings, device-specific optimizations, or content access controls.

This topic focuses specifically on implementing manifest filtering features. Before implementing manifest filtering, you must complete the basic content delivery network integration setup. If you haven't set up your basic MediaPackage and content delivery network integration yet, start with [Integrate MediaTailor with MediaPackage and CDN](#).

Manifest filtering capabilities

Before implementing manifest filtering, understand what you can accomplish with this feature:

Core filtering capabilities

Manifest filtering provides several key capabilities that help you control content delivery:

- Restrict viewer access to premium content (such as 4K HEVC)
- Target specific device types with appropriate streams
- Filter content based on audio sample rates, languages, or video codecs
- Deliver different quality tiers to different subscribers

Common use cases

These use cases demonstrate how manifest filtering can address specific business requirements:

Subscription tiers

Offer basic subscribers lower resolution streams while providing premium subscribers access to 4K content

Example: Basic tier limited to 720p, Premium tier gets up to 4K

Device optimization

Automatically serve appropriate streams based on device capabilities

Example: Mobile devices get lower bitrates, smart TVs get higher quality

Bandwidth management

Limit stream quality during peak usage periods to manage network costs

Example: Reduce maximum bitrate during high-traffic events

Regional content

Serve different audio languages or content variants based on viewer location

Example: Automatically filter for local language audio tracks

For more information about manifest filtering concepts, see [Manifest filtering](#) in the AWS Elemental MediaPackage user guide.

Configure your CDN for manifest filtering

CDN configuration for manifest filtering is essential because your CDN must forward the `aws.manifestfilter` query parameter to MediaPackage for filtering to work. Without proper query string forwarding, filter parameters will be stripped by the CDN, and all viewers will receive unfiltered manifests regardless of their subscription tier or device capabilities. This configuration ensures that your filtering logic reaches MediaPackage and functions as intended.

To enable manifest filtering through your CDN, you need to configure query string forwarding:

1. In your CloudFront distribution, create or edit the cache behavior for manifest requests.
2. For **Cache policy**, create a new policy or edit an existing one.
3. Under **Cache key settings**, choose "Include specified query strings."
4. Add `aws.manifestfilter` to the list of allowed query strings.
5. If you're also using other MediaPackage features, add their query parameters:

- `start` and `end` - For time-shifted viewing
- `time_delay` - For time delay functionality
- `_HLS_msn` and `_HLS_part` - For LL-HLS

For more information about creating distributions, see [Create a distribution](#) in the Amazon CloudFront developer guide.

Implement client-side filtering

Client-side implementation is where you define how your video players and applications request filtered content. This configuration determines what content each viewer receives based on their subscription level, device capabilities, or other criteria. Proper implementation ensures that viewers receive only the content they should have access to, while maintaining optimal CDN cache efficiency.

To implement manifest filtering in your video players and applications:

How filtering works

The filtering process works as follows:

1. Your video player or application requests a manifest URL that includes filter parameters
2. The CDN forwards the request (including query parameters) to MediaTailor
3. MediaTailor passes the filter parameters to MediaPackage when requesting the origin manifest
4. MediaPackage applies the filters and returns a customized manifest containing only the variants that match your criteria
5. MediaTailor processes the filtered manifest for ad insertion and returns it to the player

URL format for filtering

Understanding the correct URL format is critical for successful filtering implementation. Incorrect URL formatting will result in filtering parameters being ignored or causing HTTP errors. The URL structure must include filter parameters as query strings that your CDN forwards to MediaPackage. Follow these steps to implement proper URL formatting:

To implement manifest filtering in your video players:

1. Modify your player's manifest request URLs to include the appropriate filter parameters.

2. Use the following URL format with query parameters:

```
https://CloudFront-Domain/v1/master/MediaTailor-Config/index.m3u8?  
aws.manifestfilter=video_codec:h264;audio_language:en-US
```

3. When your player requests this URL, MediaTailor will pass these parameters to MediaPackage, resulting in a filtered manifest.

Common filtering scenarios

Use these examples to implement common filtering scenarios:

Device-specific content delivery

To filter based on device capabilities, add this parameter to your manifest request:

```
aws.manifestfilter=video_codec:h264;audio_sample_rate:0-44100
```

This example limits content to H.264 video and audio with sample rates up to 44.1 kHz, suitable for mobile devices.

Premium content restriction

To limit access to high-bitrate content, add this parameter to your manifest request:

```
aws.manifestfilter=video_bitrate:0-9000000
```

This example restricts video bitrates to 9 Mbps or lower, suitable for basic subscription tiers.

Language selection

To filter for specific audio languages, add this parameter to your manifest request:

```
aws.manifestfilter=audio_language:fr,en-US,de
```

This example includes only French, US English, and German audio tracks.

Resolution targeting

To filter for specific video resolutions, add this parameter to your manifest request:

```
aws.manifestfilter=video_height:240-360,720-1080
```

This example includes video streams with heights between 240-360 pixels and 720-1080 pixels, excluding mid-range resolutions.

Codec-based filtering

To filter for specific video codecs, add this parameter to your manifest request:

```
aws.manifestfilter=video_codec:h264,h265
```

This example includes only H.264 and H.265 video streams, excluding other codecs.

Special considerations and limitations

To avoid common issues when implementing manifest filtering:

Technical limitations

- For TS manifests, use audio rendition groups to avoid removing video streams that are multiplexed with filtered-out audio streams
- In TS and CMAF manifests, audio sample rate and video bitrate are not easily visible in the manifest for verification
- Request parameters appended to media playlists or segments will result in an HTTP 400 error

Error conditions

- If filtering results in an empty manifest (no streams meet the filter criteria), MediaPackage will return an HTTP 400 error
- Conflicting filter configurations (endpoint filters + query parameters) result in HTTP 404 errors
- Invalid filter syntax or unsupported filter types result in HTTP 400 errors

Performance considerations

- Each unique filter combination creates a separate cache entry, potentially reducing cache efficiency
- Complex filters with many criteria might impact manifest generation performance
- Consider using endpoint-level filters for static filtering scenarios to improve cache performance

Test your filtering implementation

Testing your manifest filtering implementation is crucial to ensure that viewers receive the correct content based on their access level and device capabilities. Failed filtering can result in viewers receiving incorrect quality levels, unsupported formats, or content they shouldn't have access to. Comprehensive testing helps identify and resolve these issues before they affect your viewers.

To verify that your manifest filtering is working correctly:

1. Request manifests with different filter parameters and verify the results
2. Check that filtered manifests contain only the expected streams
3. Test edge cases (empty results, invalid filters) to ensure proper error handling
4. Verify that your CDN is properly forwarding filter parameters
5. Test with different devices and players to ensure compatibility

For troubleshooting filtering issues, see *Troubleshoot MediaPackage CDN integration issues*.

If you encounter HTTP 400 errors, empty manifests, or filtering parameters that don't work as expected, see [Troubleshoot MediaPackage, CDN, and MediaTailor integrations](#) for specific manifest filtering troubleshooting guidance.

Optimize CDN caching for MediaTailor and MediaPackage content delivery

AWS Elemental MediaTailor requires proper content delivery network (CDN) caching configuration for optimal performance when using AWS Elemental MediaPackage as your content origin. MediaPackage provides specific cache-control headers that tell your content delivery network how long to cache different types of content. Following these recommendations ensures smooth playback and efficient content delivery.

This topic focuses specifically on optimizing caching behavior to maximize performance and minimize costs. Before implementing advanced caching optimization, ensure you have completed the basic content delivery network integration setup. If you haven't set up your basic integration yet, start with [Integrate MediaTailor with MediaPackage and CDN](#).

MediaPackage cache-control headers

MediaPackage sets specific TTL values for different content types to optimize caching behavior:

Multivariant playlists (HLS and LL-HLS)

TTL: Half the duration of the media segments

Reason: These playlists change as new segments become available, so they need frequent updates

Media playlists (regular HLS)

TTL: Half the duration of the media segments

Reason: Similar to multivariant playlists, these update as content progresses

Media playlists (LL-HLS)

TTL: 1 second

Reason: Low-latency streaming requires very frequent updates

TS media segments and init segments

TTL: 1209600 seconds (14 days)

Reason: Media segments don't change once created, so they can be cached for extended periods

CMAF media segments and initialization segments

TTL: 1209600 seconds (14 days)

Reason: Like TS segments, these are immutable once created

For comprehensive TTL recommendations across all MediaTailor workflows and additional caching optimization strategies, see [Caching optimization for CDN and MediaTailor integrations](#).

Configure CDN cache policies

Proper cache policy configuration is essential for optimal performance and cost efficiency. Different types of content (manifests, segments, initialization files) have different caching requirements. Using separate cache behaviors allows you to optimize caching for each content type, improving cache hit ratios and reducing origin load. Without proper cache policies, you might experience unnecessary origin requests, increased costs, and poor playback performance.

To properly honor MediaPackage cache-control headers and optimize caching:

1. Open your CloudFront distribution settings in the CloudFront console.

2. Create separate cache behaviors for different content types:

- Manifest requests (*.m3u8, *.mpd)
- Media segments (*.ts, *.mp4, *.m4s)
- Initialization segments

3. For each cache behavior, create or select a cache policy with these settings:

- Enable "Origin Cache-Control Headers" option
- Set "Origin request policy" to forward necessary headers
- Configure query string forwarding based on content type

Manifest cache behavior

For manifest requests (*.m3u8, *.mpd):

- **Path pattern:** *.m3u8 and *.mpd
- **Cache policy:** Honor origin cache-control headers
- **Query strings:** Forward specific parameters (see [Optimize query string forwarding](#))
- **Headers:** Forward all headers (for minimum requirements, see [Required headers for MediaTailor CDN integration](#))

Media segment cache behavior

For media segments (*.ts, *.mp4, *.m4s):

- **Path pattern:** *.ts, *.mp4, *.m4s
- **Cache policy:** Honor origin cache-control headers (14-day TTL)
- **Query strings:** None (segments don't use query parameters)
- **Compression:** Enable for improved delivery performance

Optimize query string forwarding

Query string optimization is critical for cache efficiency because unnecessary query parameters create multiple cache variations for the same content. Each unique query parameter combination creates a separate cache entry, which reduces cache hit ratios and increases origin requests. By forwarding only the query strings that MediaPackage actually uses, you maximize cache efficiency while maintaining full functionality.

Configure your CDN to forward only the query strings that MediaPackage uses, improving cache efficiency:

Essential query strings

`start` and `end` - For time-shifted viewing windows

`time_delay` - For applying time delay on manifest contents

`_HLS_msn`, `_HLS_m`, and `_HLS_part` - For LL-HLS playback requests

Feature-specific query strings

`aws.manifestfilter` - For [manifest filtering](#)

Important

Do not include any other query strings in your cache key. MediaPackage ignores unrecognized parameters, and including them reduces cache efficiency by creating unnecessary cache variations.

Performance optimization techniques

These optimizations are configured on your CDN (such as CloudFront), not in MediaPackage or MediaTailor. Implement these additional optimizations to maximize cache performance:

Origin shield

Origin shield provides an additional caching layer between your CDN edge locations and MediaPackage endpoints. This reduces the number of requests that reach your MediaPackage endpoints, which can improve performance and reduce costs, especially during traffic spikes or when cache hit ratios are lower than optimal. Origin shield is particularly beneficial for live streaming where multiple edge locations might request the same content simultaneously.

Enable origin shield to reduce load on your MediaPackage endpoints:

1. In your CloudFront distribution, enable Origin Shield for your MediaPackage origin.
2. Select an origin shield region close to your MediaPackage endpoint.
3. This creates an additional caching layer that reduces requests to MediaPackage.

Compression configuration

Enable compression for text-based responses:

- Enable compression for manifest files (*.m3u8, *.mpd)
- Do not compress media segments (already compressed)
- Ensure all headers are forwarded to MediaPackage (for minimum requirements, see [Required headers for MediaTailor CDN integration](#))

Monitor cache performance

Track these key metrics to ensure optimal cache performance:

Cache hit ratio

Target: 90% or greater for media segments, 70% or greater for manifests

Low ratios might indicate incorrect TTL settings or unnecessary query parameters

Origin request volume

Monitor requests reaching MediaPackage endpoints

High volumes might indicate caching issues

Cache key variations

Review cache key patterns to identify unnecessary variations

Too many variations reduce cache efficiency

After implementing these cache optimizations, set up monitoring to track their effectiveness. For guidance on monitoring cache hit ratios, origin request patterns, and other key performance metrics, see [Monitor performance for MediaPackage, CDN, and MediaTailor integrations](#). If you observe poor cache performance or unexpected origin requests, see [Troubleshoot MediaPackage, CDN, and MediaTailor integrations](#) for troubleshooting steps.

Monitor performance for MediaPackage, CDN, and MediaTailor integrations

AWS Elemental MediaTailor requires effective monitoring to maintain optimal performance of your AWS Elemental MediaPackage and content delivery network (CDN) integration. This topic provides

guidance on key metrics to track, monitoring tools to use, and how to set up alerts for proactive issue detection.

Before setting up monitoring, ensure your basic integration is working correctly. If you haven't completed your basic content delivery network integration setup, start with [Integrate MediaTailor with MediaPackage and CDN](#). If you need to troubleshoot issues identified through monitoring, see [CDN integration troubleshooting](#).

Key performance metrics

Monitor these essential metrics to ensure optimal performance of your MediaPackage and CDN integration:

CDN performance metrics

For comprehensive CDN performance metrics including cache hit ratio targets, origin request volume monitoring, and response time benchmarks, see [Performance benchmarks for CDN and MediaTailor integrations](#) in the CDN optimization guide.

Key EMP-specific considerations for CDN metrics:

EMP cache-control headers

What to verify: Ensure your CDN honors EMP's cache-control headers for optimal TTL behavior

Expected behavior: Different content types should have different cache durations based on EMP's headers

For detailed guidance on EMP cache optimization, see [Optimize CDN caching for MediaTailor and MediaPackage content delivery](#).

Query parameter impact

What to monitor: Track how EMP-specific query parameters affect cache efficiency

Optimization target: Ensure only necessary EMP query parameters are included in cache keys

CDN response times

What to monitor: Track response times for different content types (manifests vs. segments).

Target values:

- Cached content: less than 100ms

- Origin requests: less than 500ms

MediaPackage performance metrics

Error rates

What to monitor: Monitor HTTP error rates from both your CDN and MediaPackage endpoints. Pay particular attention to 4xx errors, which might indicate configuration issues.

Key error codes:

- 400 errors: Often related to manifest filtering issues
- 404 errors: Might indicate routing or configuration problems
- 504 errors: Timeout issues, especially with LL-HLS

Request volume and patterns

What to monitor: Track request patterns to MediaPackage endpoints to identify usage trends and capacity needs.

Patterns to watch:

- Peak usage times
- Geographic distribution of requests
- Content type distribution (live vs. on-demand)

Latency metrics

End-to-end latency

What to monitor: For LL-HLS implementations, monitor end-to-end latency from content ingest to viewer playback. High latency might indicate CDN configuration issues.

Target values:

- LL-HLS: less than 3 seconds glass-to-glass latency
- Regular HLS: less than 30 seconds

Manifest generation time

What to monitor: Time taken by MediaPackage to generate manifests, especially with filtering applied.

Target values: less than 200ms for manifest generation

Monitoring tools and setup

Setting up comprehensive monitoring tools is essential for maintaining optimal performance and quickly identifying issues before they impact viewers. Without proper monitoring, performance degradation, cache inefficiencies, or integration problems might go unnoticed until viewers experience poor playback quality. The right monitoring setup provides visibility into all aspects of your MediaPackage and CDN integration.

Use these AWS services and tools to monitor your MediaPackage and CDN integration:

Amazon CloudWatch

Amazon CloudWatch provides the foundation for monitoring your MediaPackage and CDN integration by collecting and storing metrics from both services. Proper CloudWatch configuration ensures you have the data needed to identify performance trends, troubleshoot issues, and optimize your integration. Without CloudWatch metrics, you'll lack visibility into system performance and might not detect issues until they become critical.

Set up CloudWatch monitoring for comprehensive metrics collection:

1. **MediaPackage metrics:** Enable CloudWatch metrics for your MediaPackage endpoints to track request volumes, error rates, and response times.
2. **CDN metrics:** Configure CloudWatch to collect CloudFront metrics including cache hit ratios, origin request counts, and error rates.
3. **Custom metrics:** Create custom metrics for business-specific KPIs like viewer engagement or content popularity.

CloudWatch dashboards

Create comprehensive dashboards to visualize your metrics:

1. **Overview dashboard:** High-level metrics showing overall system health
2. **CDN performance dashboard:** Detailed CDN metrics including cache performance and geographic distribution
3. **MediaPackage performance dashboard:** MediaPackage-specific metrics including request patterns and error rates

4. **Latency dashboard:** End-to-end latency metrics for different content types and regions

Log analysis

Set up log analysis for detailed troubleshooting:

1. **CDN access logs:** Enable and analyze CDN access logs to understand request patterns and identify issues
2. **MediaPackage CloudWatch logs:** Monitor MediaPackage logs for errors and performance issues
3. **Log aggregation:** Use Amazon CloudWatch Logs Insights or third-party tools to analyze log patterns

Set up alerts and notifications

Alert configuration is crucial for proactive issue detection and resolution. Without proper alerts, issues might go unnoticed until they significantly impact viewer experience or cause service disruptions. Well-configured alerts help you identify and address problems before they affect your viewers, and ensure your team is notified of critical issues that require immediate attention.

Configure proactive alerts to identify issues before they impact viewers:

Critical alerts

Set up immediate alerts for critical issues:

- **High error rates:** Alert when 4xx or 5xx error rates exceed 5% over a 5-minute period
- **Cache hit ratio drops:** Alert when cache hit ratio falls below 70% for manifests or 85% for segments
- **High latency:** Alert when end-to-end latency exceeds target thresholds
- **Origin request spikes:** Alert when origin requests increase by more than 50% compared to baseline

Warning alerts

Set up warning alerts for trends that might indicate developing issues:

- **Gradual performance degradation:** Alert when response times increase by 20% over a 30-minute period

- **Cache efficiency trends:** Alert when cache hit ratios show declining trends over time
- **Unusual traffic patterns:** Alert for unexpected changes in request volume or geographic distribution

Use monitoring data for optimization

Leverage monitoring data to continuously improve performance:

Regular performance reviews

1. **Weekly reviews:** Analyze weekly performance trends and identify optimization opportunities
2. **Monthly capacity planning:** Use traffic patterns to plan for capacity needs and CDN optimization
3. **Quarterly architecture reviews:** Evaluate overall architecture efficiency and identify areas for improvement

Common optimization actions

Based on monitoring data, consider these optimization actions:

- **Cache policy adjustments:** Modify TTL values based on actual content update patterns. For detailed TTL optimization guidance, see [Caching optimization for CDN and MediaTailor integrations](#).
- **Geographic optimization:** Add CDN edge locations in regions with high traffic
- **Query parameter optimization:** Remove unnecessary query parameters that fragment cache
- **Origin shield configuration:** Implement origin shield in regions with high origin request volumes

For detailed monitoring guidance specific to MediaPackage, see [Monitoring MediaPackage](#) in the MediaPackage user guide.

Troubleshoot MediaPackage, CDN, and MediaTailor integrations

AWS Elemental MediaTailor integration with AWS Elemental MediaPackage and content delivery network (CDN) can encounter common issues that affect playback, caching, or other integration functionality. Use this guide when you encounter playback problems, caching issues, or other integration-related errors.

For comprehensive CDN troubleshooting guidance including universal cache performance issues, HTTP error resolution, testing procedures, and diagnostic techniques that apply to all MediaTailor implementations, see [Troubleshoot CDN integration](#). This section focuses on MediaPackage specific troubleshooting requirements.

Before troubleshooting, ensure you've completed the basic integration setup correctly. If you haven't set up your integration yet or need to review the setup steps, see [Integrate MediaTailor with MediaPackage and CDN](#). For guidance on optimizing cache performance after resolving issues, see [CDN caching](#).

Manifest filtering errors

Issues with MediaPackage manifest filtering functionality, based on documented error conditions:

HTTP 400 errors with manifest filtering

Symptoms: Requests with `aws.manifestfilter` parameters return HTTP 400 Bad Request

Validated causes (from AWS documentation):

- Filter criteria result in an empty manifest (no streams match the filter conditions)
- Invalid filter parameter names or values
- Malformed query string syntax
- Duplicate or repeated filter parameters
- Filter parameter string exceeds 1024 characters
- Query parameters applied to media playlists or segments (not supported)

Solutions:

1. Review your filter parameters to ensure they match available content streams. If filtering results in no matching streams, MediaPackage returns HTTP 400.
2. Validate filter syntax against supported parameter names and value formats.
3. Check for duplicate parameters in your query string.
4. Ensure filter parameters are only applied to multivariant playlists, not to media playlists or segments.
5. Verify that your total parameter string is under 1024 characters.

Reference: [AWS Elemental MediaPackage manifest filtering error conditions](#)

Manifest filtering not working (HTTP 200 but no filtering applied)

Symptoms: Requests return HTTP 200 but manifest contains all streams instead of filtered subset

Possible causes:

- CDN not forwarding `aws.manifestfilter` query parameter to MediaPackage
- Filter parameter not found in available streams (returns unfiltered manifest with HTTP 200)

Solutions:

1. Verify that your CDN cache policy includes `aws.manifestfilter` in the list of forwarded query strings.
2. Test filter parameters directly against MediaPackage endpoints (bypassing CDN) to verify they work as expected.
3. Check that filter values match the actual characteristics of your content streams.

Reference: [AWS Elemental MediaPackage manifest filtering error conditions](#)

Diagnostic procedures

Systematic diagnostic procedures help you identify the root cause of integration issues quickly and efficiently. Following a structured approach prevents wasted time on incorrect assumptions and ensures you address the actual problem rather than symptoms. These evidence-based diagnostic steps are designed to isolate issues and guide you to the appropriate solution.

Follow these evidence-based diagnostic steps to identify issues:

Analyze cache performance

Cache performance analysis is crucial for EMP integrations because poor cache efficiency leads to increased origin load, higher costs, and potential playback issues.

For comprehensive cache performance troubleshooting including cache hit ratio analysis, cache key optimization, and systematic diagnostic steps, see [CDN cache performance issues](#) in the main CDN troubleshooting guide.

EMP-specific cache considerations:

- **EMP cache-control headers:** Verify that your CDN honors EMP's cache-control headers rather than overriding them

- **EMP query parameters:** Ensure only necessary EMP query parameters are included in cache keys
- **EMP TTL behavior:** Confirm different EMP content types have appropriate cache durations

For detailed guidance on optimizing EMP cache policies and TTL settings, see [Optimize CDN caching for MediaTailor and MediaPackage content delivery](#).

Validate manifest filtering configuration

Manifest filtering validation is essential because filtering issues can result in viewers receiving incorrect content, unsupported formats, or content they shouldn't have access to. Systematic testing helps identify whether issues are related to CDN configuration, filter parameter syntax, or content availability.

Test manifest filtering functionality systematically:

1. Test filter parameters directly against MediaPackage endpoints (bypassing CDN) to verify they work correctly.
2. Compare filtered and unfiltered manifests to confirm expected streams are included/excluded.
3. Verify that your CDN cache policy forwards the `aws.manifestfilter` query parameter.
4. Check for HTTP 400 errors and match them against the documented error conditions.

If you need to implement or modify manifest filtering after resolving issues, see [Set up manifest filtering with MediaTailor, MediaPackage, and CDN](#) for complete setup guidance.

Validate query parameter configuration

Ensure your CDN forwards only the required query parameters:

1. Review your CDN cache policy to confirm it includes only AWS recommended parameters:
 - `aws.manifestfilter` - for manifest filtering
 - `aws.manifestsettings` - for time-shifted viewing
 - `_HLS_msn` and `_HLS_part` - for LL-HLS support
2. Remove any other query parameters from your cache key, as MediaPackage ignores them and they reduce cache efficiency.

Reference: [Working with AWS Elemental MediaPackage and CDNs](#)

Error code reference

Reference for documented error conditions and their causes:

HTTP 400 Bad Request (Manifest Filtering)

Documented causes:

- Application of filter results in empty manifest
- Invalid parameter names or values
- Malformed query string syntax
- Duplicate filter parameters
- Parameter string exceeds 1024 characters
- Query parameters on media playlists or segments

Reference: [MediaPackage manifest filtering error conditions](#)

HTTP 200 OK (No Filtering Applied)

Documented causes:

- Filter parameter not found in available streams (returns unfiltered manifest)
- Only subtitle streams present after filtering (returns unfiltered manifest)

Reference: [MediaPackage manifest filtering error conditions](#)

Additional troubleshooting resources

For issues not covered in this topic, consult these official AWS resources:

- [Previewing a manifest from AWS Elemental MediaPackage](#) - Use manifest preview to troubleshoot content packaging issues
- [Increase CloudFront cache hit ratio](#) - Comprehensive guide to CDN cache optimization
- [Manifest filtering](#) - Complete guide to MediaPackage filtering functionality

Integrating AWS Elemental MediaTailor with Amazon CloudFront

AWS Elemental MediaTailor integrates with Amazon CloudFront to improve content delivery performance and reliability. CloudFront is a content delivery network (CDN) that distributes your content through a worldwide network of data centers called edge locations. When viewers request your content from MediaTailor, CloudFront routes requests to the nearest edge location. This approach reduces latency and improves performance for your viewers.

Integrating MediaTailor with CloudFront provides several key benefits:

- Reduced latency for viewers accessing personalized content
- Improved scalability for handling large audience sizes
- Enhanced reliability through redundant delivery paths
- Cost optimization through efficient caching strategies
- Advanced features like multi-Region failover with Media Quality-Aware Resiliency (MQAR)

For comprehensive information about CloudFront features, see the [CloudFront Developer Guide](#). For information about CloudFront pricing, see [CloudFront Pricing](#).

For brevity, we sometimes use "manifests" to refer collectively to multivariant playlists, media playlists, and MPDs.

Topics

- [Set up basic CloudFront integration with MediaTailor](#)
- [Optimizing MediaTailor performance with CloudFront features](#)
- [Implement multi-Region resilience for MediaTailor with MQAR](#)
- [Monitor and troubleshoot your CloudFront and MediaTailor integration](#)

Set up basic CloudFront integration with MediaTailor

AWS Elemental MediaTailor integration with Amazon CloudFront improves content delivery performance for your viewers. This topic guides you through setting up a basic CloudFront distribution for MediaTailor. With this integration, your viewers can access personalized content

through the CloudFront network. You'll also learn how to configure proper caching for different content types.

For information about passing query parameters through CloudFront for authorization and routing, see [MediaTailor manifest query parameters](#). For advanced routing using dynamic variables, see [MediaTailor domain variables for multiple content sources](#).

Prerequisites

Before configuring CloudFront with MediaTailor, ensure you have the following:

- An active AWS account with permissions to create and manage CloudFront distributions
- A configured MediaTailor playback configuration (see [Using AWS Elemental MediaTailor to insert ads](#))
- Your content origin server properly set up and accessible
- Basic understanding of video streaming concepts (HLS/DASH)

Configuring CloudFront distribution

Follow these steps to create and configure a CloudFront distribution for MediaTailor:

To create a CloudFront distribution for MediaTailor

1. Sign in to the AWS Management Console and open the CloudFront console at <https://console.aws.amazon.com/cloudfront/v3/home>.
2. Choose **Create Distribution**.
3. For **Origin domain**, enter your MediaTailor endpoint URL (for example, `a1b2c3d4.mediatailor.us-west-2.amazonaws.com`).
4. For **Protocol**, select **HTTPS only**.
5. For **Name**, enter a name that helps you identify this origin (for example, `mediatailor-origin`).
6. Configure the default cache behavior settings:
 - a. For **Path pattern**, use the default value (*).
 - b. For **Compress objects automatically**, select **Yes**.
 - c. For **Viewer protocol policy**, select **Redirect HTTP to HTTPS**.
 - d. For **Allowed HTTP methods**, select **GET, HEAD**.

- e. For **Cache policy**, select **CachingDisabled**.
- f. For **Origin request policy**, select **AllViewer** to forward all headers for the default behavior.

 Note

The default behavior uses AllViewer to safely handle any content that doesn't match specific path patterns. Specific cache behaviors for manifests and segments will be configured separately with appropriate policies.

7. Configure the distribution settings:
 - a. For **Price class**, select the option that best matches your audience locations.
 - b. For **AWS WAF web ACL**, select an existing web ACL or leave as **Do not enable security protections**.
 - c. For **Default root object**, leave empty.
 - d. For **Standard logging**, select **On** to enable logging.
8. Choose **Create Distribution**.

Configuring cache behaviors

After creating your distribution, you need to configure additional cache behaviors to handle different types of content appropriately. This section covers basic cache behavior setup for CloudFront.

For comprehensive caching optimization including advanced TTL settings, cache key configurations, and performance tuning, see [Caching optimization for CDN and MediaTailor integrations](#) in the CDN optimization guide.

Configuring manifest cache behavior

Do not cache manifests because they contain personalized content. Follow these steps to configure the cache behavior:

To configure manifest cache behavior

1. In the CloudFront console, select your distribution.
2. Choose the **Behaviors** tab.

3. Choose **Create behavior**.
4. For **Path pattern**, enter `*.m3u8` to match HLS multivariant and media playlists.
5. For **Origin**, select your MediaTailor origin.
6. For **Cache policy**, select **CachingDisabled**.
7. For **Origin request policy**, select **AllViewer** to forward all required headers for dynamic content.
8. Choose **Create**.
9. Repeat these steps for DASH manifests using the path pattern `*.mpd` to match MPDs.

This configuration ensures that each viewer receives a personalized manifest with their specific ad content. The CDN doesn't cache these manifests, so each request goes directly to MediaTailor.

Configuring segment cache behavior

Configure separate cache behaviors for ad segments and content segments to optimize performance and ensure proper CORS handling.

Configuring ad segment cache behavior

Ad segments served through the `/tm/*` path pattern require specific configuration to handle CORS properly. Follow these steps:

To configure ad segment cache behavior

1. In the CloudFront console, select your distribution.
2. Choose the **Behaviors** tab.
3. Choose **Create behavior**.
4. For **Path pattern**, enter `/tm/*` to match ad segments served by MediaTailor.
5. For **Origin**, select your MediaTailor segments origin (using the `segments.mediatailor.region.amazonaws.com` hostname).
6. For **Cache policy**, select **CachingOptimized**.
7. For **Origin request policy**, select **None**.
8. For **Response headers policy**, select **CORS-with-preflight-and-SecurityHeadersPolicy** to ensure proper CORS headers are included in responses.
9. Choose **Create**.

Configuring content segment cache behavior

Content segments can use standard caching policies for optimal performance. Configure separate behaviors for different segment formats:

To configure content segment cache behavior

1. In the CloudFront console, select your distribution.
2. Choose the **Behaviors** tab.
3. Choose **Create behavior**.
4. For **Path pattern**, enter `*.ts` to match HLS content segments.
5. For **Origin**, select your content origin.
6. For **Cache policy**, select **CachingOptimized**.
7. For **Origin request policy**, select **None**.
8. For **Response headers policy**, select **CORS-with-preflight-and-SecurityHeadersPolicy** to ensure consistent CORS handling across all content types.
9. Choose **Create**.
10. Repeat these steps for other content segment formats using appropriate path patterns:
 - `*.mp4` for MP4 segments
 - `*.m4s` for DASH segments
 - `*.cmfv` and `*.cmfa` for CMAF segments

This configuration ensures that ad segments and content segments are cached appropriately with proper CORS handling. Ad segments use the MediaTailor segments origin with CORS protection, while content segments use your content origin with optimized caching policies.

Updating MediaTailor configuration

After setting up your CloudFront distribution, update your MediaTailor configuration to use the CloudFront domain:

To update your MediaTailor configuration

1. Open the [MediaTailor console](#).
2. Select the configuration you want to update.

3. In the **CDN configuration** section, enter your CloudFront distribution domain name (for example, `d1234abcdef.cloudfront.net`) in the **CDN content segment prefix** field.
4. Save your changes.

With this configuration, MediaTailor generates manifests with URLs that point to your CloudFront distribution instead of directly to the origin.

Testing your integration

After configuring your CloudFront distribution and updating your MediaTailor configuration, test the integration:

To test your CloudFront and MediaTailor integration

1. Request a manifest through your CloudFront distribution (for example, `https://d1234abcdef.cloudfront.net/v1/master/12345/my-config/index.m3u8`).
2. Verify that the manifest contains segment URLs that point to your CloudFront domain.
3. Play the content through a video player and verify that both content and ads play correctly.
4. Check CloudFront logs to ensure requests are being routed correctly.

Example configuration

Here's an example of a CloudFront distribution configuration for MediaTailor with proper cache behaviors:

Example CloudFront distribution configuration example

```
{
  "DefaultCacheBehavior": {
    "TargetOriginId": "mediatailor-origin",
    "ViewerProtocolPolicy": "redirect-to-https",
    "AllowedMethods": {
      "Quantity": 2,
      "Items": ["GET", "HEAD"]
    },
    "CachePolicyId": "4135ea2d-6df8-44a3-9df3-4b5a84be39ad",
    "OriginRequestPolicyId": "59781a5b-3903-41f3-afcb-af62929ccde1",
    "Comment": "Default behavior with CachingDisabled and AllViewer"
```

```

},
"CacheBehaviors": [
  {
    "PathPattern": "/*.m3u8",
    "TargetOriginId": "mediatailor-origin",
    "ViewerProtocolPolicy": "redirect-to-https",
    "CachePolicyId": "4135ea2d-6df8-44a3-9df3-4b5a84be39ad",
    "OriginRequestPolicyId": "59781a5b-3903-41f3-afcb-af62929ccde1",
    "Comment": "Manifest behavior with CachingDisabled and AllViewer"
  },
  {
    "PathPattern": "/*.ts",
    "TargetOriginId": "mediatailor-origin",
    "ViewerProtocolPolicy": "redirect-to-https",
    "CachePolicyId": "658327ea-f89d-4fab-a63d-7e88639e58f6",
    "OriginRequestPolicyId": "88a5eaf4-2fd4-4709-b370-b4c650ea3fcf",
    "Comment": "Segment behavior with CachingOptimized and HostHeaderOnly"
  }
]
}

```

This example shows:

- **Default behavior:** Uses CachingDisabled and AllViewer to safely handle any content that doesn't match specific path patterns
- **Manifest behavior (*.m3u8):** Uses CachingDisabled and AllViewer for dynamic content
- **Segment behavior (*.ts):** Uses CachingOptimized and CORS-with-preflight-and-SecurityHeadersPolicy

Next steps

After setting up your basic CloudFront integration with MediaTailor, consider these next steps:

- Optimize performance with additional CloudFront features (see [Optimizing MediaTailor performance with CloudFront features](#))
- Implement multi-Region resilience with MQAR (see [Implement multi-Region resilience for MediaTailor with MQAR](#))
- Set up monitoring and troubleshooting (see [Monitor and troubleshoot your CloudFront and MediaTailor integration](#))

Optimizing MediaTailor performance with CloudFront features

AWS Elemental MediaTailor performance with Amazon CloudFront can be enhanced through additional features beyond basic configuration. After setting up your basic CloudFront configuration with MediaTailor, you can implement additional features to enhance performance, reliability, and customization options. These optimizations help deliver a better viewing experience. They also reduce costs and origin load.

For performance optimization using dynamic routing and parameter handling, see [MediaTailor dynamic ad variables for ADS requests](#). For query parameter optimization strategies, see [MediaTailor manifest query parameters](#).

Reducing origin load with Origin Shield

Origin Shield adds a caching layer between CloudFront edge locations and your origin server. This feature is valuable for live streaming and popular VOD content. It helps when many viewers request the same content at the same time.

By consolidating requests from multiple edge locations, Origin Shield reduces the load on MediaTailor and your content origins.

To enable Origin Shield for your MediaTailor origin

1. Open the CloudFront console and navigate to your distribution.
2. Select the origin that points to your MediaTailor playback configuration.
3. Under **Origin Shield**, select **Yes**.
4. From the dropdown menu, select the AWS Region that's closest to your MediaTailor endpoint.
5. Save your changes.

For high-traffic events, Origin Shield significantly reduces request load on your origin and improves reliability. For detailed instructions, see [Enabling Origin Shield](#) in the CloudFront developer guide.

Customizing content delivery with CloudFront Functions

CloudFront Functions let you run lightweight JavaScript code at the edge to modify viewer requests and responses. You can use these functions for simple customizations like URL modifications, header manipulation, or basic authentication. For MediaTailor workflows, Functions help with tasks that don't require complex processing.

CloudFront Functions provide a lightweight way to customize content delivery at the edge. Here's how to implement them for your MediaTailor integration:

To implement CloudFront Functions for MediaTailor

1. In the CloudFront console, navigate to **Functions**.
2. Create a new function and select the appropriate purpose:
 - **URL manipulation** - To modify multivariant playlist, media playlist, and MPD request URLs before they reach MediaTailor
 - **Header manipulation** - To add or modify request headers
 - **Simple authentication** - To validate tokens or query parameters
3. Write your JavaScript function code.
4. Test your function with sample MediaTailor requests.
5. Publish and associate the function with your distribution's cache behavior.

Example Sample CloudFront Function for URL normalization

```
function handler(event) {
  var request = event.request;
  var uri = request.uri;

  // Normalize URLs to lowercase to improve cache hit ratio
  if (uri.includes('.m3u8') || uri.includes('.mpd')) {
    request.uri = uri.toLowerCase();
  }

  return request;
}
```

For more information and code examples, see [Customize at the edge by using CloudFront Functions](#) in the CloudFront developer guide.

Implementing advanced customizations with Lambda@Edge

When you need more complex processing capabilities than CloudFront Functions can provide, use Lambda@Edge. This service lets you run Node.js or Python functions at CloudFront edge locations.

Lambda@Edge functions can perform sophisticated operations like complex authentication, larger response modifications, or third-party API integrations.

For more complex customizations, use Lambda@Edge functions with your MediaTailor and CloudFront integration:

To implement Lambda@Edge with MediaTailor

1. Create a Lambda function in the US East (N. Virginia) Region.
2. Write your function code for one of these use cases:
 - **URL manipulation** - To modify multivariant playlist, media playlist, and MPD request URLs before they reach MediaTailor
 - **A/B testing** - To route users to different ad decision servers
 - **Request authentication** - To add authentication headers
 - **Response header modification** - To add CORS headers
3. Publish a version of your function and create a function alias.
4. Associate the function with your CloudFront distribution at the appropriate trigger point (viewer request or viewer response).

Note

When using Lambda@Edge with MediaTailor, avoid using origin-facing triggers (origin request and origin response) if you plan to use Media Quality-Aware Resiliency (MQAR) features, as these are currently incompatible.

For more information and code examples, see [Customize at the edge by using Lambda@Edge](#) in the CloudFront developer guide.

Additional performance optimization tips

Consider these additional optimizations to further improve performance:

Optimize cache hit ratios

Monitor your cache hit ratio in CloudFront metrics and look for opportunities to improve it:

- Standardize URL patterns to improve cache key consistency
- Use query string whitelisting to include only necessary parameters in the cache key
- Consider implementing URL normalization with CloudFront Functions

Reduce latency

Implement these techniques to minimize latency:

- Enable Brotli compression for text-based responses
- Use HTTP/2 or HTTP/3 for improved connection efficiency
- Consider enabling IPv6 support for modern networks

Cost optimization

Balance performance with cost efficiency:

- Use Origin Shield to reduce redundant origin requests
- Consider price class selection based on your audience geography
- Implement aggressive caching for segments to reduce origin traffic

Next steps

After optimizing performance with CloudFront features, consider these next steps:

- Implement multi-Region resilience with MQAR (see [Implement multi-Region resilience for MediaTailor with MQAR](#))
- Set up monitoring and troubleshooting (see [Monitor and troubleshoot your CloudFront and MediaTailor integration](#))

Implement multi-Region resilience for MediaTailor with MQAR

AWS Elemental MediaTailor multi-Region resilience is enhanced through Media Quality-Aware Resiliency (MQAR), an advanced Amazon CloudFront feature that helps MediaTailor deliver the best possible streaming experience. It automatically selects the origin with the highest quality score when you have multiple origins in different AWS Regions. This feature is particularly valuable for live streaming where you need uninterrupted service.

How MQAR works

MQAR enables CloudFront to select the origin with the highest quality score automatically. This ensures that viewers receive the best possible streaming experience. When configured properly, MQAR provides these benefits:

- Automatic selection of the highest quality origin
- Seamless failover between AWS Regions during outages
- Improved viewer experience with minimal interruptions
- Real-time quality monitoring and adaptation

Step 1: Verify MQAR requirements

Before implementing MQAR, verify that your infrastructure meets these requirements. MQAR works by comparing quality scores from multiple origins, so you need identical content available in multiple AWS Regions.

- Encoders sending aligned ingest streams to all MediaPackage channels using epoch-locked CMAF ingest streamsets
- Two identical MediaPackage channels in different AWS Regions, with identical origin endpoints
- CMAF ingest for MediaPackage channels (enabled by default)
- CloudFront distribution configured to support MQAR
- MediaTailor configurations for each MediaPackage endpoint

Step 2: Configure your encoders for MQAR

Your encoders need to produce consistent, synchronized outputs across all Regions for MQAR to work effectively. This consistency allows CloudFront to make accurate quality comparisons between origins.

Configure your MediaLive outputs with these settings:

- Ensure all video framerates in the CMAF output group are consistent (all fractional or all integer framerates).
- Avoid transitions between fractional and integer framerates across encoding sessions.

- Configure output segment sequence numbers so they never go backward across encoding sessions.
- Use identical encoder output names across all Regions.

For more information about configuring MediaLive for MQAR, see [Working with MQCS](#) in the MediaLive user guide.

Step 3: Configure MediaPackage for MQAR

Set up your MediaPackage channels and endpoints with these configurations:

To configure MediaPackage for MQAR

1. Create identical channel and endpoint configurations in each AWS Region.
2. Use CMAF as the channel input type.
3. For the primary MediaPackage origin, enable **Force endpoint error** configuration with these settings:
 - Stale multivariant playlists, media playlists, or MPDs
 - Incomplete multivariant playlist, media playlist, or MPD
 - Slate input
4. For back-up MediaPackage origins, do not enable these error configurations to maximize chances of successful failover.

For more information about configuring MediaPackage for MQAR, see [Leveraging media quality scores by using AWS Elemental MediaPackage](#) in the MediaPackage user guide.

Step 4: Configure CloudFront for MQAR

In the CloudFront configuration, you enable MQAR and define how it selects between your origins. Create an origin group with the Media quality score option enabled.

To configure CloudFront for MQAR

1. Create a CloudFront distribution with origins pointing to your MediaTailor endpoints.
2. Create an origin group that includes these origins.
3. In the origin group settings, enable the **Media quality score** option.

4. Configure failover criteria to include 404 Not Found response codes. You can optionally include other 4xx/5xx codes.
5. Create separate cache behaviors for each channel's path pattern. This prevents mixing scores when the same origin group serves multiple channels.

Note

MQAR is not available when using Lambda@Edge functions in origin-facing triggers (origin request and origin response) that are associated with your distribution's cache behavior.

For more information about configuring CloudFront for MQAR, see [Media quality-aware resiliency](#) in the CloudFront developer guide.

Step 5: Configure MediaTailor for MQAR

To complete your MQAR setup, configure MediaTailor in each Region to work with your multi-Region architecture. This ensures consistent ad insertion regardless of which origin CloudFront selects.

To configure MediaTailor for MQAR

1. Create identical MediaTailor configurations in each Region, pointing to the corresponding MediaPackage endpoints.
2. Configure the CDN content segment prefix to use your CloudFront distribution domain.
3. Ensure that the ad decision server configurations are identical across Regions.

This setup ensures that regardless of which origin CloudFront selects based on quality scores, MediaTailor can continue to personalize ads consistently.

Step 6: Test your MQAR configuration

After setting up MQAR, test your configuration to ensure it works as expected:

To test your MQAR configuration

1. Request content through your CloudFront distribution.
2. Monitor real-time logs to verify that CloudFront is selecting origins based on quality scores.

3. Simulate a failure in your primary AWS Region to test failover behavior.
4. Verify that ad insertion continues to work correctly during failover.

Next steps

After implementing MQAR, consider these next steps:

- Set up monitoring and troubleshooting for your MQAR configuration (see [Monitor and troubleshoot your CloudFront and MediaTailor integration](#))
- Implement automated deployment using AWS CloudFormation (see [Automate MediaTailor and CDN with CloudFormation](#))

Monitor and troubleshoot your CloudFront and MediaTailor integration

AWS Elemental MediaTailor integration with Amazon CloudFront requires ongoing monitoring and troubleshooting to maintain optimal performance. After implementing your CloudFront and MediaTailor integration, monitor performance and prepare to troubleshoot any issues. The content delivery network (CDN) provides tools to help you understand how your distribution performs and identify potential problems.

Setting up monitoring for your integration

Implement these monitoring strategies to track the performance of your CloudFront and MediaTailor integration:

To set up monitoring for your CloudFront and MediaTailor integration

1. Enable CloudFront standard logging:
 - a. In the CloudFront console, select your distribution.
 - b. Choose the **Logs** tab.
 - c. Under **Standard logs**, choose **Edit**.
 - d. Select **On** and configure an Amazon S3 bucket for log storage.
 - e. Choose **Save changes**.
2. Set up CloudFront real-time logs:
 - a. In the CloudFront console, select your distribution.

- b. Choose the **Logs** tab.
 - c. Under **Real-time logs**, choose **Edit**.
 - d. Select **On** and configure an Amazon Kinesis Data Streams or Amazon Data Firehose delivery stream.
 - e. Include these fields in your log configuration:
 - `time-to-first-byte` - Response time
 - `sc-status` - HTTP status code
 - `c-ip` - Viewer IP address
 - `cs-uri-stem` - Request URI path
 - `cs-user-agent` - User agent
 - `x-edge-result-type` - Result type (Hit, Miss, Error)
 - f. Choose **Save changes**.
3. Create CloudWatch dashboards:
 - a. In the CloudWatch console, choose **Dashboards**.
 - b. Choose **Create dashboard**.
 - c. Add widgets for these CloudFront metrics:
 - Requests
 - BytesDownloaded
 - 4xxErrorRate
 - 5xxErrorRate
 - TotalErrorRate
 - CacheHitRate
4. Set up CloudWatch alarms:
 - a. In the CloudWatch console, choose **Alarms**.
 - b. Choose **Create alarm**.
 - c. Create alarms for these conditions:
 - 5xxErrorRate > 1% for 5 minutes
 - 4xxErrorRate greater than 5% for 5 minutes

- CacheHitRate less than 80% for 30 minutes

Monitoring MQAR performance

When using MQAR, monitoring helps you understand how CloudFront selects between your origins and whether the quality scores meet expectations. Real-time logs show these decisions as they happen.

To monitor MQAR performance

1. Enable real-time logs for your CloudFront distribution.
2. Include these fields in your log configuration:
 - `r-host` - The hostname of the selected origin
 - `sr-reason` - The reason for origin selection
 - `x-edge-mqcs` - The media quality confidence score
3. Set up a log destination in Amazon Kinesis Data Streams or Amazon Data Firehose.
4. Create dashboards or alerts based on these metrics to monitor quality scores and origin selection decisions.

Example Sample CloudWatch dashboard for MQAR monitoring

Create a CloudWatch dashboard with these metrics:

- Origin selection counts by Region
- Average quality scores over time
- Failover events
- 4xx and 5xx error rates by origin

For more information about setting up real-time logs, see [Real-time logs](#) in the CloudFront developer guide.

Troubleshooting common issues

Issues can arise with your CloudFront and MediaTailor integration even with careful planning. Understanding common problems and solutions helps you resolve issues quickly and minimize viewer impact.

If you encounter issues with your CloudFront and MediaTailor integration, check these common problems and solutions:

Manifest caching issues

Symptom: Stale manifests or ads not updating

Solution: Verify you're using the `CachingDisabled` cache policy for multivariant playlist, media playlist, and MPD paths. Check that your configuration forwards query parameters correctly.

CORS errors

Symptom: Browser console shows CORS errors when accessing content

Solution: Configure a response headers policy with appropriate CORS headers and associate it with your cache behavior.

MQAR not working

Symptom: Origin selection ignores quality scores

Solution: Check that you've enabled the Media quality score option in your origin group settings. Verify you're not using `Lambda@Edge` origin-facing triggers.

Ad insertion failures

Symptom: Ads not inserting properly

Solution: Verify your MediaTailor configuration points to your CloudFront distribution for content segment prefix. Check that your setup forwards all required headers.

For more complex issues, you can use these troubleshooting approaches:

1. Check CloudFront distribution logs for error patterns
2. Use browser developer tools to inspect network requests

3. Compare manifest content from MediaTailor directly compared to content delivered through CloudFront
4. Test with a simple player that supports detailed logging

For more troubleshooting assistance, see the [Troubleshooting](#) section in the MediaTailor user guide.

Troubleshooting workflow

Follow this systematic approach to troubleshoot issues with your CloudFront and MediaTailor integration:

To troubleshoot CloudFront and MediaTailor integration issues

1. Identify the specific issue:
 - a. Determine if the issue affects all viewers or only some
 - b. Identify which content types are affected (manifests, segments, or both)
 - c. Note any error messages or symptoms
2. Check CloudFront logs:
 - a. Look for error status codes (4xx or 5xx)
 - b. Check cache hit/miss patterns
 - c. Verify that requests are being routed to the correct origin
3. Verify configuration:
 - a. Check cache behaviors for correct path patterns
 - b. Verify cache policies are correctly applied
 - c. Confirm that origin request policies are forwarding necessary headers
4. Test direct access:
 - a. Try accessing content directly from MediaTailor (bypassing CloudFront)
 - b. Compare responses between direct access and CloudFront access
5. Implement solution:
 - a. Apply the appropriate fix based on your findings
 - b. Test to verify the issue is resolved

- c. Document the issue and solution for future reference

Next steps

After setting up monitoring and troubleshooting for your CloudFront and MediaTailor integration, consider these next steps:

- Implement automated deployment using AWS CloudFormation (see [Automate MediaTailor and CDN with CloudFormation](#))
- Create runbooks for common operational scenarios and troubleshooting procedures
- Set up automated remediation for common issues

Set up third-party CDNs for MediaTailor ad delivery

Third-party CDNs like Akamai and Fastly can significantly improve the performance and scalability of your AWS Elemental MediaTailor ad delivery while reducing bandwidth costs. However, CDN configuration for personalized advertising requires specific settings that differ from standard video delivery.

For information about passing query parameters through third-party CDNs, see [MediaTailor manifest query parameters](#). For advanced routing configurations using dynamic variables, see [MediaTailor dynamic ad variables for ADS requests](#).

This guide walks you through the complete process of setting up your third-party CDN to work optimally with MediaTailor. You'll learn how to configure two essential behaviors:

- **Manifest bypass:** Ensures each viewer receives personalized ad insertions by preventing manifest caching
- **Segment caching:** Maximizes performance and reduces costs by efficiently caching video content

The configuration process typically takes 30-60 minutes and includes provider selection, setup, verification, and optimization. Once complete, you'll have a CDN configuration that delivers personalized ads efficiently while maintaining optimal viewer experience.

Note

This guide focuses on Akamai and Fastly configurations. For CloudFront setup instructions, see [CloudFront integration](#).

Prerequisites

Before setting up your third-party CDN with MediaTailor, make sure you have:

- An active MediaTailor configuration that includes your content origin and ad decision server
- Access to your CDN's configuration interface
- A list of file extensions used in your content (.m3u8, .mpd, .ts, and so on)
- Your CDN provider's documentation for reference

For CloudFront setup instructions instead of third-party CDNs, see [CloudFront integration](#).

Terminology

To understand CDN configuration requirements, you need to know these manifest types:

- **HLS manifests:**
 - *Multivariant playlist*: The top-level manifest that contains links to media playlists
 - *Media playlist*: The second-level manifest with links to content segments
- **DASH manifests:**
 - *MPD (Media Presentation Description)*: The standard term for DASH manifests

This guide refers to all manifest files (multivariant playlists, media playlists, and MPDs) collectively as *manifests* when discussing common configuration requirements.

For general CDN configuration principles that apply to all providers, see [Set up CDN integration with MediaTailor](#).

For CDN optimization guidance, see [Performance optimization guide for CDN and MediaTailor integrations](#).

Configure CDN caching rules

CDN caching configuration is critical for MediaTailor ad delivery because it determines how your content reaches viewers. Proper configuration ensures that manifests remain personalized for each viewer while segments are efficiently cached to reduce origin load and improve performance.

This configuration typically takes 15-30 minutes per CDN provider and requires two distinct behaviors:

- **Manifest handling:** Prevents caching to ensure each viewer receives personalized ad insertions
- **Segment caching:** Maximizes cache efficiency for video content to improve delivery performance

Follow these steps to configure your CDN's caching rules for optimal ad delivery.

Choose your CDN provider from the following tabs for specific instructions:

Akamai

Configure these two behaviors in your Akamai property:

- Manifest handling to prevent caching
- Segment caching for optimal performance

Configure manifest delivery

Configure your Akamai CDN to avoid caching manifests so that each viewer receives personalized ads.

Manifest files contain the personalized ad insertion points that MediaTailor generates for each viewer. Caching these files would result in all viewers seeing identical ads, defeating the purpose of personalized advertising.

Follow these steps for manifest requests (files ending in .m3u8, .mpd, or .smil):

1. Create a behavior to match manifest file extensions (.m3u8, .mpd, .smil)
2. Set **Caching Option** to **No Store**
3. Configure cache keys to include all query parameters
4. Enable **Forward Host Header** for proper origin routing

5. Configure header forwarding for all headers. For minimum requirements, see [Required headers for MediaTailor CDN integration](#).

Configure segment delivery

Configure your Akamai CDN to cache video segments to maximize CDN efficiency and reduce origin load.

Video segments are the actual content files that can be safely cached because they don't contain personalized information. Proper segment caching reduces bandwidth costs and improves playback performance for viewers.

Follow these steps for segment requests (files ending in .ts, .mp4, .m4s, and so on):

1. Create a behavior to match segment file extensions (.ts, .mp4, .m4s)
2. Set **Honor Origin Cache Control** to **Yes**
3. Configure default time-to-live (TTL) settings for when origin headers are missing:
 - Default TTL: 86400 seconds (24 hours)
 - Maximum TTL: 604800 seconds (7 days)

Note

After configuring these behaviors, activate your property changes in Akamai Control Center.

The changes take effect after activation.

Fastly

Create these two configurations in your Fastly service:

- Manifest handling to prevent caching
- Segment caching for optimal performance

Configure manifest delivery

Configure your Fastly CDN to bypass caching for manifest files so that each viewer receives personalized ad content.

Manifest files must reach MediaTailor for each request to ensure proper ad personalization. Bypassing the cache for these files ensures that each viewer's unique targeting parameters are processed correctly.

Follow these steps for manifest requests:

1. Create a request condition to identify manifest paths
2. Set cache condition to **Do not cache** for these requests
3. Configure **Forward** settings to include all query parameters
4. Add User-Agent to your header forwarding configuration

Configure segment delivery

Configure your Fastly CDN to cache video segments to improve delivery performance and reduce origin traffic.

Segment caching is essential for cost-effective delivery and optimal viewer experience. These files are identical for all viewers and benefit significantly from CDN caching.

Follow these steps for segment requests:

1. Create a request condition to identify segment paths
2. Set **Cache Settings** to **Honor origin cache headers**
3. Configure default time-to-live (TTL) to 86400 seconds (24 hours) for when origin headers are missing

Note

After making these changes, activate a new version of your Fastly service. The configuration takes effect after activation.

Verify your CDN configuration

Verification ensures that your CDN configuration works correctly before you direct production traffic through it. These tests confirm that ad personalization functions properly and that caching provides the expected performance benefits.

Complete verification typically takes 10-15 minutes and should be performed from multiple geographic locations if possible.

After setting up your CDN, perform these checks to verify it's working correctly:

1. Test manifest personalization:
 - a. Request the same content URL with different ad parameters
 - b. Verify that each request returns different ad insertions
2. Test segment caching:
 - a. Check CDN metrics for segment cache hit ratio (should be greater than 90%)
 - b. Monitor origin traffic to confirm it's lower than direct delivery
3. Test playback performance:
 - a. Play content through your CDN from different locations
 - b. Verify smooth playback with no buffering issues

For comprehensive testing methodologies and advanced validation procedures, see [Testing and validation for CDN and MediaTailor integrations](#).

Optimize CDN performance

After verifying your basic configuration, implement these optimizations to maximize performance and minimize costs:

Monitor key performance metrics

Track these metrics to ensure optimal performance:

Cache hit ratio

Target: Greater than 90% for video segments

Impact: Higher ratios reduce origin load and improve viewer experience

Monitor: Check your CDN provider's analytics dashboard daily

Origin response time

Target: Less than 200ms for manifest requests

Impact: Faster manifest delivery reduces startup time for viewers

Monitor: Set up alerts for response times exceeding 500ms

Error rates

Target: Less than 0.1% for all requests

Impact: High error rates indicate configuration issues or origin problems

Monitor: Set up alerts for error rates exceeding 1%

Fine-tune caching behavior

Adjust these settings based on your content characteristics and viewer patterns:

Segment TTL optimization

Live content: Use shorter TTL (1-4 hours) to ensure timely updates

VOD content: Use longer TTL (24-48 hours) to maximize cache efficiency

Ad segments: Consider shorter TTL (30 minutes to 2 hours) for frequently updated ad content

For comprehensive TTL recommendations and caching strategies across all MediaTailor workflows, see [Caching optimization for CDN and MediaTailor integrations](#).

Geographic optimization

Multi-region origins: Configure origin selection based on viewer location

Edge locations: Enable additional edge locations in regions with high viewer concentration

Failover: Configure backup origins for high availability

Optimize costs

Implement these strategies to reduce CDN costs while maintaining performance:

- **Compression:** Enable gzip compression for manifest files to reduce bandwidth usage
- **Purging strategy:** Implement selective cache purging instead of full cache clears

- **Traffic analysis:** Review traffic patterns monthly to identify optimization opportunities
- **Tier selection:** Use appropriate CDN service tiers based on your performance requirements

Troubleshoot third-party CDN issues

CDN configuration issues typically manifest as either ad personalization problems or performance degradation. Use this systematic approach to identify and resolve the most common issues that affect MediaTailor ad delivery.

Most troubleshooting can be completed within 15-30 minutes by checking the specific symptoms and applying the corresponding solutions.

If viewers experience problems with ad delivery or playback quality, use this guide to identify and resolve common CDN configuration issues:

Akamai: Cached manifests

Symptom: Viewers see identical ads even when you configure different targeting parameters.

Solution: Verify that you applied the **No Store** caching option to manifest paths.

Also verify that you included query parameters in the cache key.

Fastly: Incorrect cache keys

Symptom: Viewers experience inconsistent ad personalization.

Viewers might also see ads intended for other viewers.

Solution: Verify that you configured the **Forward** settings to include all query parameters in the cache key.

General: High origin traffic

Symptom: Your origin servers experience unexpectedly high traffic

Solution: Verify segment caching settings and time-to-live (TTL) values.

Check cache hit ratios in your CDN metrics.

General: Playback errors

Symptom: Viewers experience buffering or playback failures

Solution: Check CDN routing rules and origin health.

Verify that all required headers are being forwarded correctly.

 Note

If these solutions don't resolve your issue, check your CDN provider's documentation. You can also contact their support team for additional troubleshooting steps.

For general CDN troubleshooting guidance, see [Troubleshoot issues with MediaTailor and CDN integration](#).

Performance optimization guide for CDN and MediaTailor integrations

AWS Elemental MediaTailor performance can be maximized through systematic content delivery network (CDN) optimization. Whether you're implementing server-side ad insertion (SSAI), channel assembly, or combined workflows, the optimization principles and performance targets remain consistent. This guide provides comprehensive optimization techniques and benchmarks that apply across all MediaTailor implementations.

For advanced routing optimization using dynamic variables and configuration aliases, see [MediaTailor dynamic ad variables for ADS requests](#). For query parameter optimization strategies, see [MediaTailor manifest query parameters](#).

Optimization workflow overview:

1. **Configure caching** - Set appropriate TTL values and cache behaviors
2. **Optimize routing** - Configure request routing and origin policies
3. **Measure performance** - Track against established benchmarks
4. **Apply advanced techniques** - Implement additional optimization features

Topics

- [Caching optimization for CDN and MediaTailor integrations](#)

- [Request routing optimization for CDN and MediaTailor integrations](#)
- [Performance benchmarks for CDN and MediaTailor integrations](#)
- [Advanced optimization techniques for CDN and MediaTailor integrations](#)

Caching optimization for CDN and MediaTailor integrations

AWS Elemental MediaTailor caching requirements vary by workflow type and content format. Proper caching configuration is critical for optimal performance, cost efficiency, and viewer experience. The following sections provide detailed caching guidance for different MediaTailor implementations.

Server-side ad insertion (SSAI) caching

For server-side ad insertion workflows, personalized manifests cannot be cached, but content and ad segments should be cached aggressively:

SSAI CDN caching settings

Content type	TTL	Path pattern	Cache key elements
Multivariant playlists	0 seconds	/v1/master/*	URL path + all query parameters
Media playlists	0 seconds	/v1/manifest/*	URL path + all query parameters
DASH MPDs	0 seconds	/v1/dash/*	URL path + all query parameters
Content segments	24+ hours	Content-specific paths	URL path only
Ad segments	24+ hours	/v1/segment/*	URL path only

- Set TTL of 0 seconds for personalized manifests to ensure viewers receive up-to-date ad content
- Configure longer TTL values for content and ad segments to maximize cache efficiency
- Set up cache behaviors that include personalization parameters in the cache key if you support targeted advertising

- Implement request collapsing at the CDN level to efficiently handle concurrent requests

Recommended TTL configuration settings

For optimal SSAI performance, configure your CDN cache policies with these specific TTL settings:

SSAI TTL configuration settings

Content type	TTL setting	Recommended value
Ad segments	Min TTL	1 second
Ad segments	Max TTL	86400 seconds (24 hours)
Ad segments	Default TTL	86400 seconds (24 hours)
Content segments	Min TTL	1 second
Content segments	Max TTL	86400 seconds (24 hours)
Content segments	Default TTL	86400 seconds (24 hours)

These settings ensure:

- **Min TTL of 1 second:** Allows for rapid cache invalidation when needed while preventing excessive origin requests
- **Max TTL of 24 hours:** Balances cache efficiency with content freshness requirements
- **Default TTL of 24 hours:** Provides optimal caching for segments that don't have explicit cache-control headers

Server-guided ad insertion (SGAI) caching

Server-guided ad insertion (SGAI) enables efficient CDN caching through cacheable media manifests that use predictable URL patterns. This section focuses on CDN-specific configuration requirements for optimal SGAI performance.

CDN caching configuration for SGAI

Configure your CDN with these SGAI-specific caching behaviors:

SGAI CDN caching settings

Content type	TTL	Path pattern	Cache key elements
SGAI multivariant playlists (do not cache)	0 seconds (do not cache)	/v1/master/*	URL path + selected query parameters
SGAI media playlists	1-4 seconds (half segment length)	/v1/i-media/*	URL path + selected query parameters
Asset list responses (do not cache)	0 seconds (do not cache)	/v1/interstitials/*	URL path + all query parameters
Ad segments	24+ hours	Ad-specific paths	URL path only

Cache behavior configuration

Set up dedicated cache behaviors for SGAi content:

- **SGAI manifest behavior** - Create a cache behavior for /v1/i-media/* paths with 1-4 second TTL
- **Asset list behavior** - Create a cache behavior for /v1/interstitials/* paths with 0 second TTL
- **Query parameter handling** - Include only essential targeting parameters in cache keys to maximize cache efficiency
- **Origin request headers** - Forward necessary headers for ad targeting while maintaining cacheability

Channel assembly caching

For channel assembly workflows, manifests can be cached for short periods, while segments should be cached aggressively:

Channel assembly CDN caching settings

Content type	VOD TTL	Live TTL	Path pattern	Cache key elements
Multivariant playlists	5-30 minutes	5-10 seconds	Channel-specific paths	URL path + all query parameters
Media playlists	5-30 minutes	2-5 seconds	Channel-specific paths	URL path + all query parameters
DASH MPDs	5-30 minutes	5-10 seconds	Channel-specific paths	URL path + all query parameters
Content segments	24+ hours	5-15 minutes	Content-specific paths	URL path only
Ad segments	24+ hours	24+ hours	Ad-specific paths	URL path only

- Set short TTL values for manifests to ensure viewers receive up-to-date programming
- Configure longer TTL values for content segments to maximize cache efficiency
- Set up cache behaviors that include time-shift parameters in the cache key if you support time-shifted viewing
- Include query parameters in the cache key to properly handle time-shifted viewing requests

For detailed TTL configuration settings and best practices, see [the section called “Caching optimization”](#).

Combined SSAI and channel assembly caching

When implementing both channel assembly and SSAI, ensure your caching strategy is consistent for both services to avoid conflicts and optimize performance:

Combined workflow caching settings comparison

Content type	Channel assembly	SSAI	Combined recommendation
VOD manifests	5-30 minutes	0 seconds	(use a separate config)
Live manifests	2-10 seconds	0 seconds	(use a separate config)
SGAI VOD manifests	5-30 minutes	5-30 minutes	5-30 minutes (cacheable manifests)
SGAI Live manifests	2-4 seconds	2-4 seconds	2-4 seconds (cacheable manifests)
Content segments	24+ hours	24+ hours	24+ hours (consistent)
Ad segments	24+ hours	24+ hours	24+ hours (consistent)

This configuration maximizes cache efficiency while ensuring viewers receive up-to-date manifests for personalized ad insertion.

Request routing optimization for CDN and MediaTailor integrations

Implement these routing optimizations for all AWS Elemental MediaTailor CDN integrations:

- Create separate cache behaviors for manifest and segment requests
- Configure origin request policies to control header forwarding
- Set up proper error handling and failover mechanisms
- Implement origin shields if available in your CDN to reduce origin load
- Implement request collapsing at the CDN level to efficiently handle concurrent requests

Performance benchmarks for CDN and MediaTailor integrations

When optimizing your AWS Elemental MediaTailor CDN integration, aim for these performance benchmarks. These targets apply to all MediaTailor implementations including SSAI, channel assembly, and combined workflows:

Cache Hit Ratio Targets

Content segments: greater than 95% cache hit ratio

Ad segments: greater than 90% cache hit ratio

Manifests: Not applicable for SSAI (should not be cached for personalized ad insertion); 85%+ for channel assembly

Latency Benchmarks

Manifest request latency: less than 100ms (P95)

Content segment delivery: less than 50ms (P95)

Ad segment delivery: less than 75ms (P95)

End-to-end startup time: less than 2 seconds

Origin Load Metrics

Origin requests per viewer: less than 0.1 requests per minute per viewer

Origin bandwidth per viewer: less than 5% of total viewer bandwidth

Error Rate Targets

Manifest errors: less than 0.1%

Segment errors: less than 0.01%

Player-reported rebuffering: less than 1%

Scalability Benchmarks

Support for 10 times the normal traffic during peak events without degradation

Ability to handle greater than 1000 requests per second per channel

Use Amazon CloudWatch metrics to track these performance indicators. For detailed monitoring instructions, see [Set up monitoring tools](#).

Advanced optimization techniques for CDN and MediaTailor integrations

After implementing basic caching and routing optimizations, consider these advanced techniques to further enhance performance:

Origin Shield implementation

Origin Shield adds a caching layer between CDN edge locations and your origin server, reducing origin load and improving performance:

- Enable Origin Shield for high-traffic content and live streaming
- Choose Origin Shield locations close to your MediaTailor regions
- Monitor Origin Shield cache hit ratios and adjust as needed
- Consider multiple Origin Shield locations for global deployments

Content compression optimization

Optimize content compression to reduce bandwidth and improve delivery speed:

- Enable gzip compression for manifest files
- Configure Accept-Encoding header forwarding for MediaTailor manifest compression
- Use Brotli compression where supported for additional bandwidth savings
- Avoid compressing already-compressed video segments

Regional optimization strategies

Optimize performance for global audiences through regional strategies:

- Deploy MediaTailor configurations in multiple regions for global audiences
- Use geo-routing to direct viewers to the nearest MediaTailor region
- Configure regional failover for high availability
- Monitor regional performance metrics separately

Architecture considerations for CDN and MediaTailor integrations

Position your content delivery network (CDN) correctly in your architecture to ensure optimal performance and reliability with AWS Elemental MediaTailor. The recommended architecture places the CDN between viewers and MediaTailor, not between MediaTailor and your origin.

For detailed architecture diagrams and workflow explanations, see the following topics.

- [Ad insertion with CDN](#) for ad insertion architecture diagrams and detailed workflow
- [Understand CDN architecture](#) for channel assembly architecture diagrams and workflow

Position your CDN correctly in your architecture:

1. Place your CDN between players and MediaTailor (not between MediaTailor and your origin).

This architecture allows your CDN to cache ad segments and content segments. At the same time, MediaTailor can generate personalized manifests for each viewer.

2. Create separate cache behaviors for different request types:

- Manifest requests (no caching)
- Content segments (longer TTL)
- Ad segments (longer TTL)

3. Configure proper error handling:

- Set up negative caching (temporarily storing error responses) to avoid overwhelming your origin with repeated requests during service disruptions. Negative caching means the CDN will temporarily store error responses (like 404 or 500 errors) to prevent repeated requests for content that doesn't exist or is temporarily unavailable.
- Configure appropriate error response codes and retry behavior

4. Implement intermediate caching (origin shield):

Origin shield is a feature that creates an additional caching layer between CDN edge locations and your origin server. This reduces the number of redundant requests that reach your origin server.

- Configure an intermediate caching layer between edge locations and your origin
- Reduce the number of redundant requests to your origin during cache misses
- Improve cache hit ratios across your CDN infrastructure

Advanced CDN features for MediaTailor

After implementing basic content delivery network (CDN) configuration, explore these advanced features to further enhance your AWS Elemental MediaTailor streaming platform's performance and reliability.

Media Quality-Aware Routing (MQAR)

MQAR is an Amazon CloudFront feature that automatically selects the highest quality content source based on real-time network performance metrics. Instead of using a fixed origin server, MQAR dynamically routes requests to the best-performing origin based on factors like latency and throughput. This helps ensure viewers receive the highest possible quality stream even during network fluctuations.

If you use Amazon CloudFront, implement MQAR to automatically select the highest quality content source based on real-time metrics. For details, see [CloudFront integration](#) in the CloudFront integration section.

Manifest filtering

Manifest filtering allows you to customize which renditions (different quality versions of the same content) are included in the manifests that MediaTailor delivers to viewers. Filtering helps optimize bandwidth usage by removing renditions that aren't appropriate for certain devices or network conditions. For example, you might remove 4K renditions for mobile devices or low-bandwidth connections.

For detailed information about implementing manifest filtering with AWS Elemental MediaPackage, see [MediaPackage CDN integration](#).

Multi-CDN strategy

A multi-CDN strategy uses multiple CDN providers simultaneously to improve reliability and performance. If one CDN experiences issues, traffic can automatically shift to another provider. This approach is particularly valuable for high-profile live events where reliability is critical.

For information about implementing a multi-CDN strategy with MediaTailor, see [Plan CDN integration](#).

Monitor MediaTailor CDN operations and performance

Effective monitoring of your AWS Elemental MediaTailor and content delivery network (CDN) integration ensures reliable content delivery, optimal performance, and quick issue detection. This monitoring approach applies to all MediaTailor implementations including server-side ad insertion (SSAI), channel assembly, and combined workflows.

Monitoring your CDN integration enables you to:

- Detect and resolve issues before they impact viewers
- Track key performance indicators and maintain service quality
- Maintain optimal cache performance and reduce origin load
- Ensure ad insertion success rates meet business requirements

For troubleshooting parameter-related issues that may appear in monitoring data, see [MediaTailor parameter troubleshooting guide](#). For information about monitoring query parameter usage, see [MediaTailor manifest query parameters](#).

Essential CDN performance metrics

Track these core metrics to evaluate your CDN's effectiveness with MediaTailor implementations:

Cache hit ratio

What it measures: The percentage of requests served from the CDN cache versus from origin.

Target values:

- Content segments: 95% or higher cache hit ratio
- Ad segments: 90% or higher cache hit ratio
- Manifests: Varies by implementation (personalized manifests should not be cached)

Why it matters: Higher cache hit ratios reduce origin load, improve response times, and lower bandwidth costs.

Origin request volume

What it measures: The number of requests reaching your MediaTailor origin servers.

Target pattern: Should remain low and stable, with occasional spikes for cache misses or new content.

Why it matters: High origin request volumes indicate caching inefficiencies and can impact MediaTailor performance.

Response latency

What it measures: How quickly the CDN responds to viewer requests.

Target values:

- Cached content: Less than 100ms (P95)
- Origin requests: Less than 500ms (P95)
- Manifest requests: Less than 100ms (P95)
- Segment requests: Less than 50ms (P95)

Why it matters: Low latency ensures smooth playback and good viewer experience.

Error rates

What it measures: The percentage of requests that result in HTTP errors.

Target values:

- 4xx errors: Less than 0.1% of total requests
- 5xx errors: Less than 0.01% of total requests
- Origin errors: Less than 0.05% of origin requests

Why it matters: High error rates indicate configuration issues or service problems that impact viewer experience.

MediaTailor specific metrics

Monitor these MediaTailor metrics alongside CDN metrics for complete visibility:

Ad fill rates

Key metrics: `Avail.FillRate` and `AdDecisionServer.FillRate`

Target values: Above 90% for both metrics

Why it matters: Directly impacts ad revenue and viewer experience

Manifest generation performance

Key metrics: `GetManifest.Latency` and `GetManifest.Errors`

Target values: Latency under 200ms, error rate under 1%

Why it matters: Affects playback startup time and reliability

Ad decision server health

Key metrics: `AdDecisionServer.Latency`, `AdDecisionServer.Errors`, and `AdDecisionServer.Timeouts`

Target values: Latency under 1000ms, error rate under 5%, minimal timeouts

Why it matters: ADS performance directly affects ad insertion success

Set up monitoring tools

Configure these tools to monitor your MediaTailor and CDN integration effectively:

Amazon CloudWatch integration

Amazon CloudWatch provides the foundation for monitoring your MediaTailor and CDN integration:

MediaTailor metrics

MediaTailor automatically publishes metrics to CloudWatch that track requests, responses, and errors. Key metrics include:

- `RequestCount`: Total number of requests to MediaTailor
- `ResponseTime`: MediaTailor response latency
- `4xxErrorCount` and `5xxErrorCount`: Error tracking

For a complete list of MediaTailor metrics, see [Monitoring AWS Elemental MediaTailor with Amazon CloudWatch metrics](#).

CDN metrics

Enable CDN metrics collection in CloudWatch including:

- Cache hit ratios for different content types
- Origin request counts and response times
- Error rates by status code

For CloudFront specific metrics, enable detailed monitoring in your distribution settings.

Dashboard configuration

Create dashboards that provide visibility into your MediaTailor and CDN performance:

1. **Create a unified dashboard:** Combine MediaTailor and CDN metrics in a single CloudWatch dashboard for complete visibility.
2. **Organize by workflow:** Group metrics by implementation type (SSAI, channel assembly, or combined workflows).
3. **Include key performance indicators:**
 - Cache hit ratio trends over time
 - Response latency percentiles (P50, P95, P99)
 - Error rate trends and spikes

For detailed dashboard creation guidance, see [CloudWatch dashboards](#).

Alert configuration

Configure alerts to detect issues before they impact viewers:

Critical alerts

Configure immediate alerts for severe issues:

- **High error rates:** Alert when 5xx errors exceed 0.1% of requests over 5 minutes
- **Origin server issues:** Alert when origin response time exceeds 2 seconds
- **Cache hit ratio drops:** Alert when cache hit ratio falls below 70% for manifests or 85% for segments

Warning alerts

Configure early warning alerts for performance degradation:

- **Latency increases:** Alert when P95 response time exceeds 200ms
- **Cache efficiency decline:** Alert when cache hit ratio drops below 90% for segments

Implementation checklist

Use this checklist to ensure comprehensive monitoring coverage:

1. Metrics collection:

- ✓ MediaTailor metrics enabled in CloudWatch
- ✓ CDN detailed monitoring enabled

2. Dashboard setup:

- ✓ Unified CloudWatch dashboard created
- ✓ Key metrics visualized with appropriate time ranges
- ✓ Dashboard shared with relevant teams

3. Alert configuration:

- ✓ Critical alerts configured with immediate notification
- ✓ Warning alerts set up for early detection
- ✓ Alert escalation procedures documented

4. Operational procedures:

- ✓ Incident response procedures documented
- ✓ Regular review schedule established
- ✓ Team training completed

Related topics

For additional guidance on specific monitoring scenarios:

- **Performance optimization:** For detailed optimization techniques based on monitoring data, see [CDN performance optimization](#).
- **Troubleshooting:** For detailed troubleshooting procedures using monitoring data, see your workflow-specific troubleshooting documentation.
- **Log analysis:** For comprehensive log analysis and monitoring, see [CDN integration log analysis and error code reference for MediaTailor](#).

Testing and validation for CDN and MediaTailor integrations

AWS Elemental MediaTailor content delivery network (CDN) integration requires thorough testing to ensure reliable ad delivery and optimal performance. Systematic testing helps identify integration issues before they impact viewers. Use this comprehensive approach to validate your content delivery network and MediaTailor integration across different scenarios, configurations, and load conditions.

Related topics:

- For operational setup including monitoring and logging, see [CDN monitoring](#)
- For troubleshooting when issues occur, see [Troubleshoot CDN integration](#)
- For performance optimization guidance, see [Performance optimization guide for CDN and MediaTailor integrations](#)

Topics

- [Testing prerequisites and setup for CDN and MediaTailor integrations](#)
- [Systematic testing methodology for CDN and MediaTailor integrations](#)
- [Pre-deployment testing checklist for CDN and MediaTailor integrations](#)
- [Testing tools and utilities reference for CDN and MediaTailor integrations](#)

Testing prerequisites and setup for CDN and MediaTailor integrations

AWS Elemental MediaTailor content delivery network (CDN) integration testing requires proper preparation and resource allocation. Before beginning systematic testing, ensure you have the necessary resources and tools in place.

Required resources:

- Test CDN distribution configured to mirror production settings
- Test MediaTailor configuration with known content and ad sources
- Test content with predictable characteristics (duration, format, ad break markers)
- Test ad decision server or mock ADS responses
- Multiple test devices and player types

Testing tools:

- `curl` for HTTP request testing
- `ffprobe` for HLS manifest validation
- `mp4box` for DASH manifest validation
- Browser developer tools for network analysis
- Video players for end-to-end testing

Systematic testing methodology for CDN and MediaTailor integrations

AWS Elemental MediaTailor content delivery network (CDN) integration testing should follow a systematic, phased approach to ensure comprehensive coverage. Follow this structured approach to comprehensively test your content delivery network and MediaTailor integration. Each phase builds on the previous one to isolate potential issues.

For additional guidance on systematic testing approaches, see [Testing for reliability](#) in the AWS Well-Architected Framework.

Phase 1: Test direct MediaTailor connectivity

Start by testing MediaTailor functionality without CDN involvement to establish a baseline.

1. Test manifest requests directly to MediaTailor endpoints:

- Test HLS multivariant playlist requests: `curl -v "https://your-emt-endpoint.mediatailor.region.amazonaws.com/v1/master/hls/config-name/master.m3u8"`
- Test DASH MPD requests: `curl -v "https://your-emt-endpoint.mediatailor.region.amazonaws.com/v1/dash/config-name/manifest.mpd"`
- Verify manifest responses contain expected ad break markers
- Check that segment URLs point to correct origins

2. Verify ad insertion works correctly:

- Test with different ad targeting parameters
- Verify ad segments are properly transcoded and available
- Check ad break timing and duration
- Test fallback behavior when ads are unavailable

3. Measure baseline performance:

- Record manifest request response times
- Measure ad decision server response times
- Test session creation and management

Success criteria: All direct MediaTailor requests return HTTP 200 responses with properly formatted manifests containing expected ad content.

Phase 2: Test basic CDN integration

Add CDN to the request path and test basic functionality.

1. Test manifest requests through CDN:
 - Configure CDN with simple routing rules
 - Test manifest requests through CDN endpoints
 - Verify CDN correctly forwards requests to MediaTailor
 - Check that manifest responses are not cached (TTL = 0)
2. Test segment routing:
 - Verify content segments route to origin server
 - Verify ad segments route to MediaTailor ad storage
 - Test segment caching behavior
3. Compare CDN vs direct performance:
 - Measure response time differences
 - Check for any content differences in responses
 - Verify error handling works correctly

Success criteria: CDN should successfully proxy requests to MediaTailor and origin servers with minimal performance impact.

Phase 3: Test query parameter forwarding

Add query parameter forwarding and test ad personalization.

1. Configure query parameter forwarding at the CDN:
 - Enable forwarding of all query parameters to MediaTailor
 - Test session initialization (session ID is automatically generated by MediaTailor upon first request)
 - Test with custom targeting parameters
2. Test ad personalization:
 - Verify different parameters produce different ad responses
 - Test parameter encoding and special characters
 - Check that parameters are correctly passed to ADS

3. Validate session management:

- Test session creation and persistence
- Verify session ID consistency across requests
- Test session expiration handling

Success criteria: Ad content varies based on query parameters, and sessions are managed correctly.

Phase 4: Test header forwarding

Add header forwarding in the CDN and test device-specific targeting.

1. Configure header forwarding for all headers. For minimum requirements, see [Required headers for MediaTailor CDN integration](#).
2. Test device targeting:
 - Test with different User-Agent strings (mobile, desktop, TV)
 - Verify device-specific ad responses
 - Test geographic targeting with different IP addresses
3. Validate CORS handling:
 - Test CORS headers for web player compatibility
 - Verify preflight OPTIONS requests work correctly
 - Test from different domains

Success criteria: Device and geographic targeting should work correctly, and web players should not encounter CORS errors.

Phase 5: Comprehensive scenario testing

Test across multiple scenarios to ensure robust operation.

1. Test with different player types:
 - HLS.js players in web browsers
 - Video.js players with HLS and DASH support
 - Native players on mobile devices
 - Smart TV and set-top box players
2. Test on different devices and platforms:

- Mobile devices (iOS, Android)
 - Desktop browsers (Chrome, Firefox, Safari, Edge)
 - Smart TVs and streaming devices
 - Different operating system versions
3. Test different content types:
- Live streaming content
 - Video on demand (VOD) content
 - Different video formats and bitrates
 - Content with different ad break patterns
4. Test ad targeting scenarios:
- Different demographic targeting parameters
 - Geographic targeting across different regions
 - Time-based targeting (different times of day)
 - Custom business logic parameters

Success criteria: All player and device combinations should work correctly with appropriate ad targeting.

Phase 6: Load and performance testing

Validate performance under realistic load conditions.

Important

Before load testing, contact [AWS Support](#): Before conducting load and performance testing, create an AWS Support ticket to notify the MediaTailor service team of your planned testing. This ensures:

- The service is prepared for your expected load levels
- Appropriate capacity is available during your testing window
- Your testing won't be mistaken for a production incident
- You receive guidance on testing best practices and limitations

Include in your support ticket: expected concurrent users, testing duration, geographic regions, and any specific scenarios you plan to test.

1. Test concurrent user scenarios:

- Simulate multiple concurrent viewers
- Test CDN scaling and cache performance
- Monitor origin server performance under load
- Verify MediaTailor can handle concurrent sessions

2. Measure performance metrics:

- Monitor response times under load
- Check cache hit ratios meet expectations (>80% for popular content)
- Measure time to first frame for different scenarios
- Track error rates during peak load

3. Test failover scenarios:

- Test behavior when ADS is unavailable
- Test origin server failover
- Verify error handling and recovery
- Test CDN edge location failover

Success criteria: System should maintain acceptable performance under expected load with graceful degradation during failures. Ensure that you contact [AWS Support](#) and they approve your load testing plan before execution.

Pre-deployment testing checklist for CDN and MediaTailor integrations

AWS Elemental MediaTailor content delivery network (CDN) integration must pass comprehensive testing before production deployment. Use this checklist before deploying configuration changes to production.

Basic functionality:

- ☐ Manifest requests return HTTP 200 responses

- ☐ Content segments load correctly
- ☐ Ad segments load correctly
- ☐ Ad breaks appear at expected times
- ☐ Playback transitions smoothly between content and ads

Configuration validation:

- ☐ Query parameters are forwarded correctly
- ☐ Required headers are forwarded correctly
- ☐ Manifest caching is disabled (TTL = 0)
- ☐ Segment caching is configured appropriately
- ☐ CORS headers are configured for web players

Cross-platform testing:

- ☐ Tested on mobile devices
- ☐ Tested on desktop browsers
- ☐ Tested with different player types
- ☐ Tested both HLS and DASH formats

Performance validation:

- ☐ AWS Support ticket created for load testing approval
- ☐ Response times meet performance targets
- ☐ Cache hit ratios are acceptable
- ☐ Error rates are within acceptable limits
- ☐ Monitoring and alerting are configured

Testing tools and utilities reference for CDN and MediaTailor integrations

AWS Elemental MediaTailor content delivery network (CDN) integration testing requires various tools for comprehensive validation and debugging. Reference guide for tools commonly used in content delivery network and MediaTailor integration testing.

curl - HTTP request testing

Test manifest requests: `curl -v "https://your-cdn-domain.com/v1/master/hls/config/master.m3u8"`

Test with headers: `curl -H "User-Agent: TestAgent/1.0" "https://your-cdn-domain.com/..."`

Test with parameters: `curl "https://your-cdn-domain.com/...?aws.sessionId=test123"`

ffprobe - HLS manifest validation

Validate HLS syntax: `ffprobe -v quiet -print_format json -show_format "https://your-cdn-domain.com/master.m3u8"`

Check segment information: `ffprobe -v quiet -show_entries packet=pts_time "segment.ts"`

mp4box - DASH manifest validation

Validate DASH MPD: `mp4box -info "https://your-cdn-domain.com/manifest.mpd"`

Check segment timing: `mp4box -info segment.m4s`

Browser developer tools

Monitor network requests in the Network tab

Check for CORS errors in the Console tab

Inspect request/response headers

Analyze timing and performance metrics

Additional resources:

- [Troubleshooting CloudFront](#) - Comprehensive CDN troubleshooting guide
- [Increase CloudFront cache hit ratio](#) - Performance optimization guidance
- [Monitor AWS resources](#) - Performance monitoring best practices

Troubleshoot issues with MediaTailor and CDN integration

This comprehensive troubleshooting guide covers common content delivery network (CDN) integration issues across all AWS Elemental MediaTailor implementations including server-side ad insertion (SSAI), channel assembly, and AWS Elemental MediaPackage integration. When your CDN and MediaTailor integration experiences problems, use this systematic diagnostic approach to quickly identify root causes and implement validated solutions.

This guide applies to all MediaTailor CDN integrations regardless of your specific workflow. For issues specific to particular services or workflows, see the related troubleshooting sections referenced at the end of this guide.

Before you begin: Have these items ready for efficient troubleshooting:

- Sample playback URLs that demonstrate the issue
- CDN access logs from the time period when issues occurred
- MediaTailor configuration name and AWS Region
- Player type and version (for example, HLS.js 1.4.0, Video.js 8.0)
- Device and browser information where issues occur

Related topics:

- For operational setup and troubleshooting preparation, see [Troubleshoot CDN integration](#)
- For log analysis and error code reference, see [CDN integration log analysis reference](#)
- For escalation and getting additional help, see [Get CDN integration support](#)

Diagnostic checklist for MediaTailor and CDN integrations

AWS Elemental MediaTailor content delivery network (CDN) integration problems can manifest in various ways. Use this checklist to quickly identify the type of issue you're experiencing:

1. Is the issue affecting all viewers or specific viewers?

- All viewers → Likely CDN or MediaTailor configuration issue
- Specific viewers → Likely personalization or targeting issue

2. Are manifests loading correctly?

- No → CDN routing or MediaTailor connectivity issue
- Yes, but wrong content → Caching or personalization issue

3. Are segments loading correctly?

- Content segments fail → Origin connectivity issue
- Ad segments fail → Ad delivery or transcoding issue

4. Are ads being inserted correctly?

- No ads appear → Check ADS connectivity and configuration
- Wrong ads appear → Check ad targeting parameters and personalization
- Ads fail to play → Check ad transcoding and segment availability

5. Is playback smooth and uninterrupted?

- Buffering issues → Check CDN cache performance and origin response times
- Playback errors → Check manifest syntax and segment availability
- Ad transition issues → Check ad break timing and segment alignment

6. Are there specific error codes or messages?

- HTTP 4xx errors → Check CDN routing and configuration
- HTTP 5xx errors → Check origin server and MediaTailor service health
- Player-specific errors → Check manifest format and player compatibility

Next steps based on your diagnosis:**CDN configuration issues**

For detailed CDN routing and caching troubleshooting, see [Troubleshoot issues with MediaTailor and CDN integration](#).

Manifest and playback issues

For manifest validation and playback troubleshooting, see [CDN integration testing procedures](#).

Ad insertion and targeting issues

For ad-specific troubleshooting including ADS connectivity and ad delivery, see your workflow-specific troubleshooting documentation.

Performance and monitoring issues

For performance analysis and monitoring setup, see [Monitor MediaTailor CDN operations and performance](#).

Log analysis and error codes

For detailed log analysis and error code reference, see [CDN integration log analysis and error code reference for MediaTailor](#).

Testing and validation

For comprehensive testing procedures, see [Testing and validation for CDN and MediaTailor integrations](#).

If you need immediate assistance or cannot resolve the issue using the linked resources, see [Get support and troubleshooting help for CDN and MediaTailor integrations](#) for escalation procedures.

CDN integration testing procedures

Proper testing is essential before deploying your MediaTailor CDN integration to production. These testing procedures help identify configuration issues, performance problems, and compatibility issues across different devices and platforms.

Basic integration validation

Perform these fundamental tests to verify your CDN integration is working correctly:

1. Test manifest delivery:

- Request a manifest through your CDN and verify it returns a valid response
- Verify the manifest contains expected content and ad insertion points
- Check that manifest URLs use your CDN domain, not the origin
- Validate manifest syntax using HLS or DASH validation tools

2. Verify URL rewriting:

- Check that content segment URLs in manifests point to your CDN domain
- Verify ad segment URLs point to your CDN domain

- Ensure all relative URLs are properly resolved

3. Test content playback:

- Play content through a video player and verify smooth playback
- Verify both content and ads play without interruption
- Check for proper transitions between content and ads
- Test seeking and scrubbing functionality

4. Validate CDN routing:

- Monitor CDN access logs to ensure requests are routed correctly
- Verify cache hit/miss patterns are as expected
- Check that origin requests only occur for cache misses

Advanced testing procedures

Perform these additional tests for comprehensive validation:

1. Cross-platform compatibility testing:

- Test on multiple devices (desktop, mobile, tablet, smart TV)
- Verify compatibility across different browsers
- Test with various video players (HLS.js, Video.js, native players)
- Validate on different operating systems

2. Performance testing:

- Measure manifest request response times (target: <100ms for cached)
- Test segment loading performance across different bitrates
- Verify startup time meets performance targets
- Test under various network conditions

3. Ad tracking validation:

- Verify ad tracking beacons fire correctly
- Check ad analytics data for accuracy
- Test impression and completion tracking
- Validate click-through functionality

4. Error condition testing:

- Test behavior when origin is temporarily unavailable

- Verify graceful handling of malformed requests
- Test CDN failover scenarios
- Validate error message clarity and usefulness

Create a test environment

Set up a test environment that mirrors your production configuration for comprehensive validation:

1. Set up separate CDN distributions for testing:
 - Create test CDN distributions with the same cache behaviors as production
 - Configure test origins that mirror your production setup
 - Use separate domain names to avoid conflicts with production traffic
2. Create test MediaTailor configurations:
 - Set up test playback configurations with the same settings as production
 - Configure test ad decision server endpoints
 - Use test ad content that matches your production ad formats
3. Implement systematic testing processes:
 - Create testing checklists for configuration changes
 - Document test procedures for your team
 - Set up automated testing where possible

Test across multiple scenarios

Validate your integration across different scenarios and conditions to ensure comprehensive coverage:

1. Test with multiple player types and devices:
 - Test with different video players (web, mobile, connected TV)
 - Validate across different operating systems and browsers
 - Test on various network conditions and connection speeds
2. Create automated testing scripts:
 - Automate manifest request validation
 - Create scripts to test ad insertion scenarios

- Implement performance testing for high-traffic scenarios
3. Validate ad targeting and personalization:
- Test with different user profiles and targeting parameters
 - Validate ad decision server integration
 - Test fallback scenarios when ads are unavailable

Testing tools and techniques

Use these tools and techniques for effective testing:

Browser developer tools

Use Network tab to inspect HTTP requests and responses

Monitor console for JavaScript errors and warnings

Verify response headers and caching behavior

Check timing information for performance analysis

Command-line testing

Use curl to test specific URLs and inspect headers:

```
curl -I "https://your-cdn-domain.com/path/to/manifest.m3u8"
```

Use wget for download testing and timing analysis

Employ tools like httpie for more readable HTTP testing

Video player testing

Test with multiple player implementations

Use player debug modes to inspect internal behavior

Monitor player events and error callbacks

Validate adaptive bitrate switching behavior

CDN analytics and monitoring

Monitor real-time CDN metrics during testing

- Review access logs for request patterns
- Use CDN-specific testing tools when available
- Set up temporary alerts for testing validation

For additional comprehensive testing methodologies and systematic validation approaches, see [Testing and validation for CDN and MediaTailor integrations](#).

Troubleshoot CDN manifest 404 errors for MediaTailor

AWS Elemental MediaTailor content delivery network (CDN) manifest 404 errors are a common integration issue that prevents playback from starting. This section provides step-by-step troubleshooting for manifest delivery failures.

Multivariant playlist, media playlist, or MPD requests return 404 errors

Quick fix (try first):

1. Verify the MediaTailor configuration name in your URL matches exactly (case-sensitive)
2. Test the manifest URL directly against MediaTailor without CDN: `curl -v "https://your-emt-endpoint.mediatailor.region.amazonaws.com/v1/master/hls/config-name/master.m3u8"`
3. If direct test works, check CDN routing rules for manifest requests

If quick fix doesn't work:

Symptoms: Players fail to start playback, manifest requests return HTTP 404 errors in CDN logs.

Example error messages:

- Browser console: "Failed to load resource: the server responded with a status of 404 (Not Found)"
- Player error: "MANIFEST_LOAD_ERROR" or "NETWORK_ERROR"
- CDN logs: `GET /v1/master/hls/example-config/master.m3u8 404`

Resolution:

Verify that your CDN routing rules are correctly configured to forward multivariant playlist, media playlist, and MPD requests to MediaTailor.

Check that the MediaTailor configuration exists and is correctly set up.

Ensure your CDN behavior patterns match the expected manifest request paths (for example, *.m3u8, *.mpd).

Diagnose CDN manifest delivery issues and errors for MediaTailor

AWS Elemental MediaTailor content delivery network (CDN) manifest delivery problems can prevent proper ad insertion and playback. If viewers receive incorrect or inconsistent ads in multivariant playlists, media playlists, or MPDs:

1. Check for cached manifests:

- Verify TTL settings are set to 0 for all multivariant playlist, media playlist, and MPD paths
- Confirm that your CDN isn't caching multivariant playlists, media playlists, or MPDs despite TTL settings
- Check CDN logs for cache status - manifest requests should show Miss or RefreshHit, not Hit

2. Verify CDN routing configuration:

- Confirm manifest requests are being routed to MediaTailor endpoints, not cached or served from origin
- Check that CDN behavior patterns correctly match manifest paths (*.m3u8, *.mpd)
- Verify query parameters are being forwarded to MediaTailor for personalization
- Test manifest URLs directly against MediaTailor to isolate CDN vs service issues

3. Check header forwarding configuration:

- Verify required headers are being forwarded (see [Required headers for MediaTailor CDN integration](#))
- Confirm User-Agent header is forwarded for device-specific ad targeting
- Check that X-Forwarded-For header is forwarded for geo-targeting
- Ensure Accept-Encoding header is forwarded for compression support

4. Validate manifest content and structure:

- Check that manifests contain expected ad insertion markers (EXT-X-CUE-OUT/IN for HLS)
- Verify segment URLs in manifests use your CDN domain, not origin domains
- Confirm ad segments are properly inserted and accessible
- Test manifest syntax using validation tools (ffprobe for HLS, mp4box for DASH)

5. Test different scenarios:

- Test with different session IDs to verify personalization is working
- Test from different geographic locations to verify geo-targeting
- Test with different User-Agent strings to verify device targeting
- Compare manifest responses with and without CDN to identify differences

Additional troubleshooting resources:

- For detailed CDN caching configuration, see [Caching optimization for CDN and MediaTailor integrations](#)
- For comprehensive CDN routing setup, see [Set up CDN routing behaviors for MediaTailor](#)
- For header forwarding requirements, see [Required headers for MediaTailor CDN integration](#)
- For log analysis and error codes, see [CDN integration log analysis and error code reference for MediaTailor](#)
- For testing procedures and validation, see [Testing and validation for CDN and MediaTailor integrations](#)

Success criteria: When resolved, players should start playback normally and ads should appear as expected. Manifest requests should return HTTP 200 status codes in CDN logs, and manifests should contain properly personalized ad content.

Troubleshoot CDN segment delivery and loading issues for MediaTailor

AWS Elemental MediaTailor content delivery network (CDN) segment delivery problems can cause buffering and playback interruptions. If players can't load segments or experience buffering:

1. Check CDN routing rules:

- Verify content segments are being routed to the correct origin
- Confirm ad segments are being routed to the correct MediaTailor ad storage location
- Check that segment file extensions match your CDN behavior patterns
- Verify that segment URLs in manifests use the correct CDN domain
- For detailed instructions on setting up routing and behavior path patterns, see [Set up CDN routing behaviors for MediaTailor](#)

2. Verify CORS configuration:

- For web players, ensure your CDN is passing through or properly setting CORS headers

- Test with browser developer tools to identify CORS-related errors
 - Verify that preflight OPTIONS requests are handled correctly
3. Test segment accessibility and performance:
- Test individual segment URLs directly to verify they're accessible
 - Check segment response times and identify performance bottlenecks
 - Verify segment file sizes are appropriate for your bandwidth targets
 - Test segment loading from different geographic locations
4. Validate CDN caching behavior for segments:
- Verify content segments have appropriate TTL settings (typically longer than manifests)
 - Check that ad segments are cached appropriately based on personalization requirements
 - Monitor cache hit ratios for both content and ad segments
 - Ensure cache keys don't include unnecessary parameters that reduce cache efficiency
5. Check origin server connectivity and health:
- Verify origin servers are responding correctly to segment requests
 - Check origin server capacity and response times under load
 - Validate that origin servers have the expected segment files available
 - Test origin failover scenarios if multiple origins are configured
6. Troubleshoot ad segment specific issues:
- Verify ad segments are properly transcoded and available in MediaTailor
 - Check that ad segment URLs are correctly generated in manifests
 - Test ad segment loading with different ad targeting parameters
 - Monitor for ad transcoding delays that might cause segment unavailability
7. Validate player compatibility and behavior:
- Test segment loading with different player types and versions
 - Check player buffer settings and segment request patterns
 - Verify player error handling for failed segment requests
 - Test adaptive bitrate switching and segment selection logic

Additional troubleshooting resources:

- For CDN routing and behavior configuration, see [Set up CDN routing behaviors for MediaTailor](#)⁶¹⁵

- For CDN caching optimization, see [Caching optimization for CDN and MediaTailor integrations](#)
- For CORS configuration guidance, see [CDN integration security best practices for MediaTailor](#)
- For performance monitoring and analysis, see [Monitor MediaTailor CDN operations and performance](#)
- For comprehensive testing procedures, see [Testing and validation for CDN and MediaTailor integrations](#)
- For log analysis and error diagnosis, see [CDN integration log analysis and error code reference for MediaTailor](#)

Success criteria: When resolved, players should load segments smoothly without buffering interruptions. Segment requests should return HTTP 200 status codes with appropriate response times, and both content and ad segments should be accessible and properly cached.

Fix CDN session management and tracking issues for MediaTailor

AWS Elemental MediaTailor content delivery network (CDN) session management is critical for proper ad personalization and tracking. If you encounter session-related errors or inconsistent behavior across requests:

1. Check for session ID consistency:

- Verify that your player maintains the same session ID across all requests for a single playback session
- Check CDN logs to confirm session IDs are being forwarded correctly
- Ensure session IDs are properly URL-encoded in query parameters
- Use CloudWatch Logs to verify session ID consistency across requests (see validation steps below)

2. Validate session initialization:

- Confirm that the first manifest request successfully creates a session
- Check for proper session parameter forwarding (for example, `aws.sessionId`)
- Verify session initialization using debug logs (see debug log setup below)

3. Enable debug logging for detailed session troubleshooting:

- **For server-side reporting:** Add `?aws.logMode=DEBUG` to your playback request:


```
GET <mediatailorURL>/v1/master/<hashed-account-id>/<origin-id>/<asset-id>?
aws.logMode=DEBUG
```

- **For client-side reporting:** Include "logMode": "DEBUG" in your session initialization request body
- **Important:** The DEBUG value is case-sensitive
- Maximum of 10 active debug sessions allowed simultaneously

4. Use CloudWatch Logs queries to validate session behavior:

- **Verify debug session is active:**

```
fields @timestamp, @message
| filter sessionId = "your-session-id-here"
| filter eventType = "SESSION_INITIALIZED" # client-side reporting
or mediaTailorPath like "/v1/master" # server-side reporting HLS
or mediaTailorPath like "/v1/dash" # server-side reporting DASH
```

- **View all events for a session:**

```
fields @timestamp, @message, eventType, mediaTailorPath
| filter sessionId = "your-session-id-here"
| sort @timestamp asc
```

- **Check manifest generation for a session:**

```
fields @timestamp, responseBody, @message
| filter mediaTailorPath like "/v1/master/" and eventType = "GENERATED_MANIFEST"
and sessionId = "your-session-id-here"
```

5. Test session parameter forwarding through CDN:

- Test manifest requests with session parameters directly against MediaTailor (bypassing CDN)
- Compare session behavior with and without CDN to identify forwarding issues
- Verify CDN query parameter forwarding configuration includes session-related parameters
- Check that CDN doesn't cache responses that should be session-specific

Common session error messages:

- **ConflictException (HTTP 409)** - Multiple simultaneous playlist requests for the same session.
Solution: Ensure your player requests playlists one at a time according to HLS specification
- **NotFoundException (HTTP 404)** - Session is unavailable or configuration doesn't exist.
Solution: Check your configuration validity and reinitialize the session
- **BadRequestException (HTTP 400)** - Invalid session ID or improperly formatted request.
Solution: Verify request formatting and session ID validity

Additional troubleshooting resources:

- For complete debug logging setup and field reference, see [Generating AWS Elemental MediaTailor debug logs](#)
- For CloudWatch Logs query examples and log analysis, see [Writing AWS Elemental MediaTailor logs directly to Amazon CloudWatch Logs](#)
- For CDN query parameter forwarding configuration, see [Set up CDN routing behaviors for MediaTailor](#)
- For comprehensive error code reference, see [Troubleshooting playback from MediaTailor](#)

Success criteria: When resolved, sessions should initialize correctly, maintain consistent session IDs across requests, and debug logs should show proper SESSION_INITIALIZED events and manifest generation without errors.

Resolve CDN ad break timing and synchronization issues for MediaTailor

AWS Elemental MediaTailor content delivery network (CDN) ad break timing must be precisely synchronized with content markers. If ads appear at incorrect times or ad break timing is inconsistent:

1. Verify ad break markers in content:
 - Check that SCTE-35 markers are properly placed in your origin content
 - Verify ad break duration matches actual ad content length
 - Confirm that ad break timing aligns with content boundaries
 - Validate SCTE-35 marker format and timing accuracy in your origin manifests
 - Test ad break markers with different content types (live vs VOD)

2. Check CDN caching impact on timing:

- Ensure manifest TTL is set to 0 to prevent timing drift
- Verify that time-sensitive parameters are not being cached
- Check for clock synchronization issues between the content source, MediaTailor, and the CDN
- Monitor for timing drift in long-running live streams
- Verify CDN edge server time synchronization with NTP

3. Validate SCTE-35 marker implementation:

- Verify EXT-X-DATERANGE tags include proper SCTE35-OUT and DURATION specifications
- Check for paired SCTE35-OUT and SCTE35-IN markers when using explicit cue-in timing
- Validate START-DATE timestamps align with actual content timing
- Test different SCTE-35 marker formats (duration-based vs paired markers)

4. Test ad break timing across different scenarios:

- Compare ad break timing with direct MediaTailor requests vs CDN requests
- Test timing consistency across different CDN edge locations
- Verify ad break timing with different player types and buffering behaviors
- Monitor timing accuracy during peak traffic periods

5. Debug timing issues using logs and monitoring:

- Enable debug logging to track ad break processing timing
- Monitor CloudWatch metrics for ad insertion timing patterns
- Check CDN logs for timing-related request patterns
- Use player debugging tools to verify ad break timing from client perspective

Expected timing tolerances:

- Ad break timing should align with SCTE-35 markers in your content
- Ad duration should match the duration specified in your ad decision server response
- Clock synchronization between content source, MediaTailor, and CDN should be within 1 second
- SCTE-35 marker timing should be accurate to within 100ms of actual content timing

Additional troubleshooting resources:

- For SCTE-35 marker format and implementation, see [Integrating a content source for MediaTailor ad insertion](#)
- For debug logging setup and timing analysis, see [Generating AWS Elemental MediaTailor debug logs](#)
- For CDN caching configuration and timing impact, see [Caching optimization for CDN and MediaTailor integrations](#)
- For comprehensive testing procedures including timing validation, see [Testing and validation for CDN and MediaTailor integrations](#)
- For monitoring ad insertion timing and performance, see [Monitor MediaTailor CDN operations and performance](#)

Success criteria: When resolved, ad breaks should appear at precisely the times specified by SCTE-35 markers, with consistent timing across all CDN edge locations and player types. Debug logs should show accurate ad break processing timing without drift or synchronization errors.

Optimize CDN performance and resolve latency issues for MediaTailor

AWS Elemental MediaTailor content delivery network (CDN) integration performance directly impacts viewer experience and ad delivery quality. If you experience slow response times or performance degradation:

Performance measurement techniques

Before troubleshooting performance issues, establish baseline measurements and ongoing monitoring:

1. Measure key performance metrics:
 - **Response times:** Manifest requests should complete within 200ms, segment requests within 100ms
 - **Cache hit ratios:** Content segments >95%, ad segments >90%
 - **Origin request volume:** Should be less than 5% of total requests when cache is optimized
 - **Time to first frame:** Initial playback should start within 2-3 seconds
2. Use performance measurement tools:
 - **CDN analytics dashboards:** Monitor cache performance, response times, and error rates
 - **CloudWatch metrics:** Track MediaTailor service metrics including GetManifest.Latency

- **Browser developer tools:** Measure client-side performance and network timing
- **Command-line tools:** Use curl with timing options to measure specific requests

3. Implement continuous monitoring:

- Set up automated performance alerts for response time degradation
- Monitor performance across different geographic regions
- Track performance during peak traffic periods
- Compare performance metrics before and after configuration changes

Performance measurement resources:

- For comprehensive performance monitoring setup, see [Monitor MediaTailor CDN operations and performance](#)
- For performance testing procedures, see [Testing and validation for CDN and MediaTailor integrations](#)
- For CloudWatch metrics and monitoring, see [Monitoring AWS Elemental MediaTailor with Amazon CloudWatch metrics](#)

CDN cache performance issues

Cache performance problems are among the most common CDN integration issues. These problems affect all MediaTailor implementations and can significantly impact viewer experience and costs.

Low cache hit ratio

Symptoms: High origin request volume, increased latency, higher bandwidth costs, poor viewer experience

Target values:

- Content segments: 95% or higher cache hit ratio
- Ad segments: 90% or higher cache hit ratio
- Manifests: Varies by implementation (personalized manifests should not be cached)

Common causes:

- Incorrect TTL settings for different content types

- Cache key configuration includes unnecessary query parameters
- Cache-control headers from origin not properly configured
- Frequent cache invalidations or purges
- Geographic distribution issues (content not cached at edge locations)

Solutions:**1. Review and optimize TTL settings:**

- Content segments: Set TTL to match segment duration or longer
- Ad segments: Set TTL to 24 hours or longer for reusable ads
- Static assets: Set TTL to 24 hours or longer

For comprehensive TTL recommendations and caching optimization strategies, see [Caching optimization for CDN and MediaTailor integrations](#).

2. Optimize cache key configuration:

- Remove unnecessary query parameters from cache keys
- Ensure only content-affecting parameters are included
- Normalize parameter order and case sensitivity

3. Verify origin cache-control headers are properly set**4. Implement origin shield (or equivalent CDN functionality) for high-traffic implementations.**

Origin shield functionality is available across major CDNs but might have different names (such as CloudFront Origin Shield, Fastly Shield, Cloudflare Argo Tiered Cache). If your CDN doesn't offer this functionality, it can be enabled in MediaTailor when you contact [AWS Support](#).

5. Review cache invalidation strategies and reduce unnecessary purges**Validation steps:**

1. Monitor cache hit ratios using CDN analytics dashboards
2. Test specific URLs with curl to verify cache headers
3. Compare origin request volume before and after changes

High origin request volume

Symptoms: Unexpectedly high number of requests reaching MediaTailor origin, increased origin server load, higher costs

Expected pattern: Origin requests should be less than 5% of total viewer requests when cache hit ratios are optimal

Common causes:

- Cache misses due to low TTL values
- Cache key fragmentation (too many unique cache keys)
- Geographic traffic spikes in regions without cached content
- Frequent cache invalidations

Solutions:

1. Analyze request patterns to identify cache miss causes
2. Optimize TTL settings based on content type and update frequency
3. Implement cache warming strategies for new content
4. Consider origin shield implementation (available across major CDNs with different names - see [Origin Shield implementation](#) for details)

Alert threshold: Set alerts when origin requests exceed 10% of total requests or increase by 50% over baseline

Common HTTP error resolution

HTTP errors in CDN integrations often indicate configuration issues or service problems. These error patterns are consistent across all MediaTailor implementations.

404 Not Found errors

Symptoms: Manifest or segment requests return HTTP 404, players fail to load content, "MANIFEST_LOAD_ERROR" in player logs

Common causes:

- Incorrect CDN origin configuration (wrong MediaTailor endpoint URL)
- Missing or incorrect cache behavior path patterns
- URL rewriting issues in CDN configuration
- MediaTailor configuration name or playback endpoint errors

- Timing issues with live content (requesting future segments)

Diagnostic steps:

1. Test the same URL directly against MediaTailor origin (bypass CDN)
2. Verify CDN origin configuration matches MediaTailor playback endpoint
3. Check CDN cache behavior path patterns and precedence
4. Review CDN access logs for request routing details
5. Validate MediaTailor configuration name and region settings

Solutions:

- Correct CDN origin configuration to match MediaTailor playback endpoint
- Update cache behavior path patterns to properly route requests
- Fix URL rewriting rules if applicable
- Verify MediaTailor configuration exists and is active

403 Forbidden errors

Symptoms: Requests return HTTP 403, access denied messages, authentication failures

Common causes:

- Missing or incorrect query parameters required by MediaTailor
- CDN not forwarding required headers or parameters
- IP address restrictions or geographic blocking
- Authentication token issues (if using signed URLs)

Solutions:

- Verify all required query parameters are included and forwarded
- Check CDN configuration for header and parameter forwarding
- Review IP restrictions and geographic settings
- Validate authentication tokens and signing processes

400 Bad Request errors

Symptoms: Requests return HTTP 400, malformed request errors, parameter validation failures

Common causes:

- Malformed query parameters or URL encoding issues
- Invalid parameter values or formats
- Missing required parameters for specific MediaTailor features
- URL length limitations exceeded

Solutions:

- Validate query parameter formats and URL encoding
- Check parameter values against MediaTailor API requirements
- Ensure all required parameters are included
- Review URL length and consider parameter optimization

5xx Server errors

Symptoms: Requests return HTTP 500, 502, 503, or 504 errors, intermittent service failures

Common causes:

- MediaTailor service issues or capacity limitations
- CDN origin connectivity problems
- Timeout issues due to slow origin responses
- Temporary service degradation

Solutions:

- Check AWS Service Health Dashboard for MediaTailor service status
- Verify CDN origin connectivity and timeout settings
- Implement retry logic with exponential backoff
- Monitor MediaTailor CloudWatch metrics for service health
- Contact AWS Support if issues persist

1. Measure baseline performance:

- Test manifest request response times directly to MediaTailor (target: <200ms)
- Measure CDN response times for manifest requests (target: <100ms for cache hits)
- Check segment loading times from both origin and CDN

2. Analyze CDN performance:

- Check cache hit ratios for content segments (target: >80% for popular content)
- Verify origin shield (or equivalent CDN functionality) is enabled and configured in the same AWS Region as your origin. Different CDNs use different names for this functionality
- Monitor CDN edge location performance and geographic distribution

Performance benchmarks:

- Monitor manifest generation response times and compare against your baseline performance
- CDN cache hits are significantly faster than origin requests
- ADS response times should not cause manifest generation delays

Additional troubleshooting resources:

- For comprehensive performance optimization strategies, see [Performance optimization guide for CDN and MediaTailor integrations](#)
- For origin shield implementation details, see [Origin Shield implementation](#)
- For CDN caching optimization, see [Caching optimization for CDN and MediaTailor integrations](#)
- For performance monitoring and metrics, see [Monitor MediaTailor CDN operations and performance](#)
- For performance testing procedures, see [Testing and validation for CDN and MediaTailor integrations](#)

Success criteria: When resolved, response times should meet target benchmarks (manifests less than 200ms, segments less than 100ms), cache hit ratios should exceed 90% for most content types, and origin request volume should be less than 5% of total requests. Performance should be consistent across all geographic regions and device types.

Fix CDN inconsistent behavior across devices and platforms for MediaTailor

AWS Elemental MediaTailor content delivery network (CDN) integration should provide consistent ad delivery across all devices and platforms. If ads behave differently across devices:

1. Ensure consistent header forwarding across all CDN behaviors.

- Verify that User-Agent, X-Forwarded-For, and custom targeting headers are forwarded consistently
 - Check that header forwarding rules apply to all relevant cache behaviors
2. Verify player compatibility with your CDN configuration.
 - Test with multiple player types (HLS.js, Video.js, native players) to identify player-specific issues
 - Check for player-specific header requirements or URL handling differences
 3. Test with multiple device types to identify device-specific issues.
 - Include mobile devices, tablets, smart TVs, and desktop browsers in your testing
 - Test different operating systems and browser versions
 - Verify that device-specific ad targeting works correctly

If you've followed these troubleshooting steps and still need assistance, see [Get CDN integration support](#).

Troubleshooting preparation

Set up tools and processes to simplify troubleshooting when CDN integration issues arise. Proactive preparation makes troubleshooting faster and more effective when issues occur.

Enable comprehensive logging

Detailed logs are essential for diagnosing CDN integration issues. Configure logging to capture the information you'll need during troubleshooting.

1. Enable detailed CDN access logs:
 - Configure logging for all cache behaviors that handle MediaTailor requests
 - Include query strings and custom headers in log entries
 - Set up log analysis tools to identify patterns and anomalies
 - Enable real-time logs for immediate issue detection during live events
 - Configure log retention policies to maintain historical troubleshooting data
2. Configure MediaTailor logging:
 - Enable access logs for your MediaTailor configurations
 - Set up CloudWatch log groups for centralized log management

- Configure log filters to identify error patterns
3. Set up origin server logging:
 - Enable detailed access logs on your content origin servers
 - Include request headers and response codes in logs
 - Monitor origin server performance metrics

Add diagnostic request headers

Custom headers help track requests through your CDN and identify routing issues.

1. Configure CDN diagnostic headers:
 - Add a unique identifier to each request (for example, X-Request-ID)
 - Include CDN-specific information in request headers
 - Add edge location or POP (point of presence) information to track geographic routing
 - Include cache status headers (Hit, Miss, RefreshHit) for cache behavior analysis
2. Add response headers for debugging:
 - Include server identification headers
 - Add timing information for performance analysis
 - Include cache control headers for manifest requests

Establish baseline performance metrics

Document normal performance ranges to quickly identify anomalies during troubleshooting:

1. **Record baseline metrics:**
 - Cache hit ratios for different content types
 - Response time percentiles (P50, P95, P99)
 - Error rates by status code
 - Request volume patterns by time of day
2. **Document performance expectations:**
 - Target cache hit ratios (95%+ for content, 90%+ for ads)
 - Acceptable response times (<100ms cached, <500ms origin)
 - Maximum acceptable error rates (<1% for 4xx, <0.1% for 5xx)

3. **Create performance dashboards:** Set up monitoring dashboards that show current metrics compared to baseline values.

Prepare troubleshooting tools

Set up tools and access permissions needed for effective troubleshooting:

1. Command-line tools:

- `curl` for testing HTTP requests and responses
- `dig` or `nslookup` for DNS troubleshooting
- HLS/DASH validation tools for manifest verification
- Log analysis tools (`grep`, `awk`, or specialized log analyzers)

2. Access permissions:

- CDN management console access for configuration review
- MediaTailor console access for configuration verification
- CloudWatch access for metrics and log analysis
- Origin server access for backend troubleshooting

3. Documentation:

- Network architecture diagrams
- CDN and MediaTailor configuration documentation
- Contact information for escalation procedures
- Troubleshooting runbooks for common scenarios

Workflow-specific troubleshooting guides

This universal troubleshooting guide covers common issues across all MediaTailor CDN integrations. For issues specific to particular workflows or services, consult these specialized troubleshooting resources:

Server-side ad insertion (SSAI) troubleshooting

For SSAI-specific issues including ad insertion failures, ad transition problems, and personalization issues, see workflow-specific SSAI troubleshooting documentation.

Common SSAI-specific issues:

- Ad insertion failures and empty ad breaks
- Ad transition timing and synchronization problems
- Personalization and targeting issues
- Ad tracking and analytics discrepancies

Channel assembly troubleshooting

For channel assembly-specific issues including manifest generation problems and time-shift functionality, see channel assembly workflow documentation.

Common channel assembly issues:

- Manifest generation and compilation errors
- Time-shift window and DVR functionality problems
- Source content availability and failover issues
- Program schedule and metadata synchronization

MediaPackage integration troubleshooting

For MediaPackage specific issues including manifest filtering and EMP endpoint problems, see [CDN integration troubleshooting](#).

Common MediaPackage integration issues:

- Manifest filtering parameter errors
- MediaPackage endpoint connectivity issues
- EMP-specific cache behavior problems
- MediaPackage origin authentication issues

CloudFront specific troubleshooting

For CloudFront specific configuration and setup issues, see CloudFront integration documentation.

Common CloudFront issues:

- Distribution configuration and cache behavior setup
- Origin access identity and security configuration
- CloudFront specific error codes and responses

- Geographic restrictions and edge location issues

Additional resources:

- For performance optimization guidance, see [CDN performance optimization](#)
- For monitoring and alerting setup, see [CDN monitoring](#)
- For general support and assistance, see [Get CDN integration support](#)

CDN integration log analysis and error code reference for MediaTailor

AWS Elemental MediaTailor content delivery network (CDN) integration logs provide valuable insights into performance and errors. This guide covers both CDN logs (from your content delivery network) and MediaTailor logs and error codes that are relevant to CDN integration troubleshooting. Use this reference when you need to understand what your content delivery network logs and error codes are telling you about your MediaTailor integration. This guide helps you interpret log entries and error messages to identify the root cause of issues.

Related topics:

- For step-by-step troubleshooting procedures, see [Troubleshoot CDN integration](#)
- For proactive monitoring and prevention strategies, see [Monitor MediaTailor CDN operations and performance](#)
- For escalation and getting additional help, see [Get CDN integration support](#)
- For comprehensive MediaTailor logging configuration and resources, see [MediaTailor logging configuration resources](#)
- For CloudFront log format reference, see [CloudFront access log format](#)
- For HTTP status code reference, see [HTTP response status codes](#)

CDN log interpretation and analysis techniques for MediaTailor

AWS Elemental MediaTailor content delivery network (CDN) integration generates detailed logs that help you understand request patterns and identify issues. When analyzing content delivery network logs, focus on these key indicators:

Note: The following examples use Amazon CloudFront terminology. Other CDN providers may use different terms for similar concepts.

HTTP status codes

- 200 - Successful request
- 404 - Resource not found (check routing rules and MediaTailor configuration)
- 403 - Access denied (check CDN security settings and origin permissions)
- 502/503/504 - Origin server errors (check MediaTailor service health and origin connectivity)

Cache behavior indicators

- Hit - Content served from CDN cache
- Miss - Content fetched from origin
- RefreshHit - Cached content validated with origin
- Error - Request resulted in an error

Request patterns to monitor

- Manifest requests should typically result in Miss or RefreshHit due to low TTL settings
- Content segments should show Hit for popular content
- Ad segments may show Miss due to personalization

Service health monitoring

When analyzing CDN logs and troubleshooting issues, check MediaTailor service health to determine if problems are service-related:

AWS Service Health Dashboard

Check current MediaTailor service status and any ongoing service events

Access: [AWS Service Health Dashboard](#)

Use this when: You see widespread 5xx errors or service timeouts in CDN logs

AWS Personal Health Dashboard

View account-specific service health notifications and maintenance events

Access: [AWS Personal Health Dashboard](#)

Use this when: You need account-specific service health information or maintenance notifications

Amazon CloudWatch metrics monitoring

Monitor MediaTailor service health indicators through CloudWatch metrics:

- `GetManifest.Errors` - Track manifest generation errors
- `GetManifest.Latency` - Monitor response time performance
- `AdDecisionServer.Errors` - Monitor ad server connectivity issues
- `Origin.Errors` - Track origin server connectivity problems

Access: [CloudWatch Console](#)

Use this when: You need detailed service performance metrics and historical trends

Service health troubleshooting workflow:

1. Check AWS Service Health Dashboard for current service status
2. Review AWS Personal Health Dashboard for account-specific notifications
3. Monitor CloudWatch metrics for service performance indicators
4. Correlate service health status with CDN log patterns and error rates
5. If service health is normal, focus on CDN configuration and origin server issues

CDN log analysis resources

- [CloudFront access logs](#) - Complete guide to CDN log format and fields
- [CloudFront real-time logs](#) - Real-time log streaming and analysis
- [Analyzing log data with CloudWatch Logs Insights](#) - Advanced log querying and analysis
- [Monitor workload resources](#) - Well-Architected monitoring patterns

Topics

- [MediaTailor error codes and CDN integration troubleshooting](#)
- [CDN log analysis tools and monitoring techniques for MediaTailor](#)
- [MediaTailor logging configuration resources](#)

MediaTailor error codes and CDN integration troubleshooting

AWS Elemental MediaTailor error codes provide specific information about integration issues when passed through your content delivery network (CDN). When MediaTailor returns errors through your content delivery network, these codes indicate specific issues:

Note: These error codes are returned by MediaTailor and passed through your CDN. For CDN-specific errors, consult your CDN provider's documentation.

400 Bad Request

Common causes: Malformed query parameters, missing required parameters, invalid session IDs

Check: CDN query string forwarding configuration, parameter encoding

403 Forbidden

Common causes: Client player requesting a segment that does not exist or lacks permission to access. For ad segments, MediaTailor might have specified a non-existent segment (contact [AWS Support](#) to investigate). For content segments, origin provider access restrictions or authentication issues

Check: For ad segments: contact [AWS Support](#) if MediaTailor is generating invalid segment URLs. For content segments: Verify origin server permissions, authentication credentials, and access policies. Check if segments exist at the requested URLs

404 Not Found

Common causes: MediaTailor configuration doesn't exist or is inactive, incorrect playback URL path, manifest or segment requests to non-existent resources

Check: Verify MediaTailor configuration exists and is active, confirm playback URL matches the `ManifestEndpointPrefix` from `GetPlaybackConfiguration`, check CDN routing rules forward requests to correct MediaTailor endpoints

500 Internal Server Error

Common causes: Origin server issues, ADS connectivity problems, manifest processing errors

Check: Origin server health, ADS response validity, MediaTailor service status

502 Bad Gateway

Common causes: Origin server unreachable, invalid origin response, `UnsupportedManifestException` due to HLS playlist alignment issues (SCTE markers not

aligned across playlists, missing SCTE markers on some playlists, inconsistent ad break timing across bitrate variants)

Check: Origin server connectivity and firewall rules, DNS resolution, HLS playlist consistency across all bitrate variants, SCTE-35 marker alignment on same segments across all playlists, verify all playlists contain required ad break markers

Error code analysis resources:

- [Troubleshooting MediaTailor](#) - Complete MediaTailor error code reference
- [Troubleshooting CloudFront error responses](#) - CDN-specific error analysis
- [HTTP response status codes](#) - Comprehensive HTTP status code reference
- [Failure management](#) - Well-Architected error handling patterns

CDN log analysis tools and monitoring techniques for MediaTailor

AWS Elemental MediaTailor content delivery network (CDN) integration generates large volumes of log data that require efficient analysis tools and techniques. Use these approaches to efficiently analyze content delivery network and MediaTailor logs:

- **Command-line analysis:** Use tools like `grep`, `awk`, and `sort` to filter and analyze log patterns
- **Amazon CloudWatch Logs Insights:** Query CDN and MediaTailor logs with SQL-like syntax for advanced analysis
- **Third-party tools:** Consider log analysis platforms for comprehensive monitoring and alerting
- **Custom dashboards:** Create visualizations that combine CDN metrics with MediaTailor performance data

If you need additional assistance with log analysis or interpreting complex error patterns, see [Get CDN integration support](#).

Log analysis tools and resources:

- [CloudWatch Logs Insights](#) - SQL-like queries for log analysis
- [Amazon OpenSearch Service](#) - Advanced log search and analytics
- [CloudWatch dashboards](#) - Custom visualization and monitoring
- [Instrumenting distributed systems for operational visibility](#) - Advanced observability patterns

- [Design workload observability](#) - Well-Architected observability guidance

MediaTailor logging configuration resources

In addition to CDN logs, MediaTailor provides comprehensive logging capabilities for monitoring ad insertion, manifest generation, and service interactions. Use these resources to configure and analyze MediaTailor logs:

MediaTailor manifest and service logs

Configure and analyze logs for manifest generation, origin interactions, and service events:

- [AWS Elemental MediaTailor manifest logs description and event types](#) - Complete reference for MediaTailor manifest logs and event types
- [Viewing AWS Elemental MediaTailor logs](#) - Guide to viewing and interpreting MediaTailor logs

Vended logs configuration

Configure flexible log delivery to multiple destinations with cost optimization:

- [Using vended logs to send AWS Elemental MediaTailor logs](#) - Configure log delivery to Amazon S3, Firehose, or CloudWatch Logs
- [Migrating your AWS Elemental MediaTailor logging strategy](#) - Migration guide for existing logging configurations

CloudWatch Logs integration

Integrate MediaTailor logs with CloudWatch for monitoring and analysis:

- [Viewing AWS Elemental MediaTailor logs](#) - CloudWatch Logs configuration and analysis
- [CloudWatchInsights](#) - Advanced log querying and analysis

Ad-specific logging

Monitor ad insertion performance and troubleshoot ad-related issues:

- ADS interaction logs - Monitor ad decision server communication and errors

Key differences between CDN and MediaTailor logs:

- **CDN logs:** Show request/response patterns, cache behavior, and network-level errors from your content delivery network

- **MediaTailor logs:** Show ad insertion details, manifest generation events, origin interactions, and service-specific errors
- **Combined analysis:** Use both log types together for complete visibility into your streaming workflow

For comprehensive monitoring that combines both CDN and MediaTailor logging, see [Monitor MediaTailor CDN operations and performance](#).

Automate MediaTailor and CDN with CloudFormation

Automating AWS Elemental MediaTailor with a content delivery network (CDN) using AWS CloudFormation streamlines your ad insertion workflow and improves reliability.

This section shows you how to use AWS CloudFormation (AWS infrastructure as code service) to automatically set up AWS Elemental MediaTailor with a content delivery network (CDN). Although you can manually configure this integration as described in previous sections, using CloudFormation saves time and reduces errors by automating the entire process with a single template.

If you're new to CloudFormation, it's a service that lets you create a template file that defines all of the AWS resources that you need. When you deploy this template, CloudFormation automatically creates and configures those resources for you, ensuring that they work together correctly.

For more information about CloudFormation, see the [CloudFormation User Guide](#). For information about MediaTailor resource types in CloudFormation, see [AWS::MediaTailor Resource Type Reference](#).

Topics

- [Why use CloudFormation for MediaTailor and CDN integration](#)
- [Prepare for CloudFormation deployment of CDN and MediaTailor integrations](#)
- [Deploy the CloudFormation template for CDN and MediaTailor integrations](#)
- [Use the CloudFormation deployed resources for CDN and MediaTailor integration](#)
- [Test and validate your CloudFormation deployment for CDN and MediaTailor integration](#)
- [Troubleshoot common CloudFormation deployment issues for CDN and MediaTailor integrations](#)
- [Customize the CloudFormation template for CDN and MediaTailor integrations](#)

- [AWS CloudFormation template reference for AWS Elemental MediaTailor and Amazon CloudFront integration](#)

Why use CloudFormation for MediaTailor and CDN integration

AWS Elemental MediaTailor automation with AWS CloudFormation provides significant advantages for broadcast professionals managing streaming workflows. Manually configuring MediaTailor with a content delivery network (CDN) can be time-consuming and error-prone. Using CloudFormation automation offers the following advantages.

- **Consistency:** Ensures the same configuration is deployed every time, reducing human error.
- **Version control:** Store your infrastructure as code in a version control system for tracking changes.
- **Rapid deployment:** Deploy complex configurations in minutes instead of hours of manual configuration.
- **Environment replication:** Easily replicate configurations across development, testing, and production environments.
- **Documentation:** The template itself serves as documentation of your infrastructure.

Here's how the automated workflow compares to manual configuration:

Manual setup (multiple steps)	Automated setup (single template)
Create MediaTailor playback configuration	Deploy one CloudFormation template with parameters
Create CloudFront distribution	
Configure cache behaviors	
Set up security configurations	

The automated workflow for setting up MediaTailor with CloudFront follows these steps:

1. Deploy the CloudFormation template with your content origin and ad server parameters
2. CloudFormation creates and configures all necessary resources:

- MediaTailor playback configuration for ad insertion
 - CloudFront distribution with appropriate cache behaviors
 - Security configurations for content protection
3. Use the CloudFormation outputs to access your ad-enabled stream URLs
 4. Stream your content with dynamically inserted ads

Prepare for CloudFormation deployment of CDN and MediaTailor integrations

AWS Elemental MediaTailor deployment with AWS CloudFormation requires specific prerequisites and preparation steps. Before you begin working with CloudFormation to integrate MediaTailor and Amazon CloudFront, make sure you have the following.

- An AWS account with permissions to create MediaTailor, CloudFront, and CloudFormation resources
- A content origin where your video content is hosted (such as AWS Elemental MediaPackage, Amazon S3, or another origin server)
- An ad decision server (ADS) that can respond to VAST requests

Before deploying the CloudFormation template, gather these required parameters:

AdServerUrl

URL of the VAST ad server for dynamic ad insertion. A static VAST endpoint is provided for testing.

ContentOriginDomainName

Domain name of your content origin without protocol (e.g., *mediapackage-domain.mediapackagev2.us-west-2.amazonaws.com*, *mybucket.s3.amazonaws.com*, or *custom-origin.example.com*). Do not include `http://` or `https://` prefixes or any paths.

ContentOriginType

The type of content origin:

- *mediapackagev2*: For AWS Elemental MediaPackage origins
- *s3*: For Amazon S3 bucket origins

- *custom*: For any other origin type

The template will create several AWS resources that work together to deliver your content with personalized ads. The following describes what each component does:

Origin access control

Origin Access Control (OAC) is a security feature that ensures your content can only be accessed through CloudFront, not directly from your origin server. This helps protect your content from unauthorized access.

For MediaPackage and Amazon S3 origins, the template creates an Origin Access Control (OAC) resource to secure access to your content.

MediaTailor playback configuration

The MediaTailor playback configuration is the core component that handles ad insertion. It receives content from your origin, requests ads from your ad server, and combines them into a personalized stream for each viewer.

The template creates an MediaTailor playback configuration with the following settings:

- Video content source pointing to your CloudFront distribution
- Ad decision server URL configured to your specified VAST endpoint
- Live pre-roll configuration for ad insertion during live streams
- CDN configuration with appropriate segment URL prefixes

CloudFront distribution

The CloudFront distribution delivers your content to viewers worldwide with low latency. It handles different types of requests (manifests, content segments, ad segments) and routes them to the appropriate origins.

For broadcast professionals new to CDNs, here are some key terms:

Origin

A server where your original content is stored (like MediaPackage or Amazon S3)

Cache behavior

Rules that determine how different types of content are cached and delivered

Cache policy

Settings that control how long content is cached and what request components affect caching

The template creates a CloudFront distribution with the following components:

- Three origins:
 - Content origin (MediaPackage, Amazon S3, or custom)
 - MediaTailor manifests origin
 - MediaTailor segments origin
- Cache behaviors with appropriate patterns:
 - Default behavior for content segments
 - Behavior for MediaTailor ad segments (/tm/*)
 - Behavior for MediaTailor interstitial media (/v1/i-media/*)
 - Behavior for personalized manifests (/v1/*)
 - Behavior for segment redirect requests (/segment/*)
- Optimized cache policies for each behavior:
 - CachingOptimized for cacheable content
 - CachingDisabled for personalized manifests
- Origin request policies to ensure proper header forwarding
- Response header policies for CORS support

Deploy the CloudFormation template for CDN and MediaTailor integrations

AWS Elemental MediaTailor deployment using the AWS CloudFormation template is straightforward once you understand what the template will create. This process takes about 15-30 minutes, with most of the time spent waiting for the CloudFront distribution to deploy.

To deploy the CloudFormation template and set up your automated ad insertion workflow:

[Deploy the template](#)

To deploy the MediaTailor CloudFormation template

1. Download the CloudFormation template from the AWS Elemental MediaTailor GitHub repository or copy it from the [AWS CloudFormation template reference for AWS Elemental MediaTailor and Amazon CloudFront integration](#).
2. Open the [CloudFormation console](#).
3. Choose **Create stack > With new resources (standard)**.
4. Under **Specify template**, choose **Upload a template file** and upload the template.
5. Enter a stack name and provide values for the required parameters:
 - **AdServerUrl**: URL of your VAST ad server (e.g., <https://your-ad-server.com/vast>)
 - **ContentOriginDomainName**: Domain name of your content origin without protocol (e.g., [mediapackage-domain.mediapackagev2.us-west-2.amazonaws.com](#))
 - **ContentOriginType**: Select the type of content origin:
 - *mediapackagev2*: For AWS Elemental MediaPackage origins
 - *s3*: For Amazon S3 bucket origins
 - *custom*: For any other origin type
6. Review the configuration and choose **Create stack**.
7. Wait for the stack creation to complete, which typically takes 5-10 minutes. You can monitor progress in the **Events** tab.
8. Once complete, navigate to the **Outputs** tab to find the URLs for your HLS and DASH manifests.

Note

If you're using AWS Elemental MediaPackage as your content origin, make sure your MediaPackage endpoints are properly configured and accessible. For more information, see [MediaPackage CDN integration](#).

Use the CloudFormation deployed resources for CDN and MediaTailor integration

AWS Elemental MediaTailor resources deployed by the AWS CloudFormation stack provide several important outputs that you'll use to access your content with ad insertion. After the CloudFormation stack is created successfully, you'll need to understand how to use the outputs to access your content with ads inserted. This is similar to how you would use MediaTailor URLs in a manual setup, but the CloudFormation deployment provides these URLs automatically.

After successful deployment, the CloudFormation stack provides several important outputs that you'll use to access your content with ad insertion:

`CloudFrontDomainName`

The domain name of your CloudFront distribution (e.g., *d1234abcdef.CloudFront.net*)

`HlsManifestUrl`

Base URL for HLS manifests with ad insertion (e.g., *https://d1234abcdef.CloudFront.net/v1/master/12345/my-playback-config/*)

`DashManifestUrl`

Base URL for DASH manifests with ad insertion (e.g., *https://d1234abcdef.CloudFront.net/v1/dash/12345/my-playback-config/*)

`MediaTailorPlaybackConfigName`

Name of the created MediaTailor playback configuration (such as *my-stack-PlaybackConfig*)

Construct playback URLs

To create the complete playback URL for your content with ads, you'll need to combine the base URL from the CloudFormation outputs with your specific manifest path. This is a critical step for broadcast professionals to understand, as it connects your existing content with the ad insertion system.

1. Start with the appropriate manifest URL from the outputs:

HlsManifestUrl: `https://d1234abcdef.CloudFront.net/v1/master/12345/my-playback-config/`

2. Append your specific manifest path:

Your manifest path: `channel/index.m3u8`

3. The complete playback URL becomes:

`https://d1234abcdef.CloudFront.net/v1/master/12345/my-playback-config/channel/index.m3u8`

Use this URL in your video player to play content with dynamically inserted ads.

Tip

If you're not sure what your manifest path should be, check your origin server. For MediaPackage origins, this is the path to your endpoint's HLS or DASH manifest. For Amazon S3 origins, this is the path to your manifest file within the bucket.

For more information about MediaTailor URL structure, see [Set up CDN integration with MediaTailor](#).

Configure a video player

After you have your playback URL, you need to configure a video player to use it. For broadcast professionals, this is similar to configuring a player for any HLS or DASH stream, but now the stream will include personalized ads. Here's a simple example using the popular HLS.js player:

```
<!DOCTYPE html>
<html>
<head>
  <title>MediaTailor Playback Example</title>
  <script src="https://cdn.jsdelivr.net/npm/hls.js@latest"></script>
</head>
<body>
  <video id="video" controls style="width: 640px; height: 360px;"></video>
```

```
<script>
    const video = document.getElementById('video');
    const mediaUrl = 'https://<replaceable>d1234abcdef.CloudFront.net</
replaceable>/v1/master/<replaceable>12345</replaceable>/<replaceable>my-playback-
config</replaceable>/<replaceable>channel/index.m3u8</replaceable>';

    if (Hls.isSupported()) {
        const hls = new Hls();
        hls.loadSource(mediaUrl);
        hls.attachMedia(video);
    } else if (video.canPlayType('application/vnd.apple.mpegurl')) {
        video.src = mediaUrl;
    }
</script>
</body>
</html>
```

You can also use professional broadcast players like:

- JW Player
- Bitmovin Player
- THEOplayer
- Video.js

For more information about player integration with MediaTailor, see [MediaTailor ad server integration requirements](#).

Test and validate your CloudFormation deployment for CDN and MediaTailor integration

AWS Elemental MediaTailor deployment validation is a critical step for broadcast professionals before going live. This section guides you through testing your deployment to ensure ads are being inserted properly and content is delivered smoothly.

After deploying the CloudFormation template, follow these steps to verify that your setup is working correctly:

To test your MediaTailor and CloudFront integration

1. Verify that all resources were created successfully in the CloudFormation console.

2. Check that the MediaTailor playback configuration is active in the [MediaTailor console](#).
3. Verify that the CloudFront distribution is deployed and enabled in the [CloudFront console](#).
4. Test playback using a sample manifest:
 1. Construct the full playback URL as described in [Construct playback URLs](#).
 2. Use a video player that supports HLS or DASH (like VLC, JW Player, or the AWS console player).
 3. Verify that content plays and ads are inserted at the expected break points.
5. Check the MediaTailor logs in CloudWatch for any ad insertion errors.

When testing ad insertion, look for these indicators of success:

- Smooth transitions between content and ads
- Ads appear at the expected break points (pre-roll, mid-roll, post-roll)
- Ad quality matches the content quality
- No buffering or playback errors during ad transitions

For more detailed testing procedures, see [Understanding AWS Elemental MediaTailor ad insertion behavior](#). For comprehensive CDN integration testing and validation, see [Testing and validation for CDN and MediaTailor integrations](#).

Troubleshoot common CloudFormation deployment issues for CDN and MediaTailor integrations

AWS Elemental MediaTailor deployment issues can occur even with automation during deployment or playback. As a broadcast professional, understanding how to troubleshoot these issues will help you maintain a reliable streaming service with ad insertion.

If you encounter issues with your CloudFormation deployment or the resulting MediaTailor and CloudFront integration, refer to these common problems and solutions:

CloudFormation deployment issues

Stack creation fails with "Resource creation failed" error

Possible causes:

- Invalid content origin domain name format
- Insufficient permissions to create resources

Solution: Check the specific resource error in the CloudFormation events tab. Verify that the content origin domain name is correctly formatted without protocol prefixes or paths. Ensure your IAM role has sufficient permissions to create all required resources.

CloudFront distribution takes a long time to deploy

Cause: CloudFront distributions typically take 15-30 minutes to fully deploy.

Solution: This is normal behavior. Wait for the distribution to reach the "Deployed" state before testing.

Playback and ad insertion issues

Content plays but no ads are inserted

Possible causes:

- Ad decision server not responding or returning empty VAST
- Content does not contain ad markers

Solution: Verify that your ad server is accessible and returning valid VAST responses. Check that your content has proper ad markers (SCTE-35 markers for live content or ad break tags for VOD).

403 Forbidden errors when accessing content

Possible causes:

- Origin access control not configured correctly
- Origin bucket or endpoint permissions issues

Solution: For Amazon S3 origins, verify that the bucket policy allows access from the CloudFront distribution. For MediaPackage origins, check that the origin access control is properly configured and that the endpoint is accessible.

Playback errors or buffering

Possible causes:

- Cache behavior path patterns not matching content paths
- Incorrect origin domain configuration

Solution: Verify that your cache behaviors have the correct path patterns to route requests to the appropriate origins. Check CloudFront logs to see which origin is handling the requests and confirm it's the expected one.

For broadcast professionals, these additional troubleshooting tips may help:

- Use the Amazon CloudWatch Logs Insights to search for specific error patterns in MediaTailor logs
- Test with a simple VAST ad server first (like the default one provided in the template) before using your production ad server
- Verify your content's ad markers using the MediaTailor manifest inspector tool in the console
- Check network traffic in your browser's developer tools to see if requests to the ad server are being made correctly

For additional troubleshooting, check the CloudWatch logs for both MediaTailor and CloudFront to identify specific errors.

Customize the CloudFormation template for CDN and MediaTailor integrations

AWS Elemental MediaTailor template customization allows broadcast professionals to adapt the AWS CloudFormation template to fit specific workflow requirements. Although the basic template works for many scenarios, these customizations can help you address more complex needs.

The examples below show YAML code snippets that you can add to the template. If you're not familiar with YAML or CloudFormation syntax, consider working with a developer or AWS solutions architect to make these changes.

You can customize the CloudFormation template to meet your specific workflow requirements.

Add or modify origins

For broadcast workflows that use multiple content sources (like primary and backup sources, or different content libraries), you can add additional origins to your CloudFront distribution:

Origins:

```
# Add a new origin for additional content
- Id: SecondaryContentOrigin
  DomainName: secondary-content.example.com
  CustomOriginConfig:
    OriginProtocolPolicy: 'https-only'
    OriginSSLProtocols:
      - TLSv1.2
```

Then add a corresponding cache behavior to route specific patterns to this origin:

CacheBehaviors:

```
- PathPattern: '/secondary-content/*'
  TargetOriginId: SecondaryContentOrigin
  ViewerProtocolPolicy: 'https-only'
  CachePolicyId: 658327ea-f89d-4fab-a63d-7e88639e58f6 # Managed-CachingOptimized
```

Create custom cache policies

For broadcast workflows with specific caching requirements (like quality selection parameters or viewer authentication), you can create custom cache policies instead of using the managed ones. For detailed guidance on TTL values and caching strategies, see [Caching optimization for CDN and MediaTailor integrations](#).

```
# Define a custom cache policy
CustomCachePolicy:
  Type: AWS::CloudFront::CachePolicy
  Properties:
    CachePolicyConfig:
      Name: !Sub '${AWS::StackName}-CustomCachePolicy'
      DefaultTTL: 86400 # 24 hours
      MaxTTL: 31536000 # 1 year
      MinTTL: 1 # 1 second
      ParametersInCacheKeyAndForwardedToOrigin:
        CookiesConfig:
          CookieBehavior: none
        HeadersConfig:
          HeaderBehavior: none
        QueryStringsConfig:
          QueryStringBehavior: whitelist
          QueryStrings:
```

- quality
- format

```
# Reference the custom policy in a cache behavior
```

```
CacheBehaviors:
```

- PathPattern: '/custom-path/*'
TargetOriginId: ContentOrigin
ViewerProtocolPolicy: 'https-only'
CachePolicyId: !Ref CustomCachePolicy

Enhance MediaTailor configuration

For broadcast workflows that need advanced ad insertion features, you can enhance the MediaTailor configuration with options like ad prefetching (to reduce latency), personalization thresholds, and bumper ads.

```
MediaTailorPlaybackConfig:
  Type: AWS::MediaTailor::PlaybackConfiguration
  Properties:
    # Add ad prefetching for improved performance
    AvailSuppression:
      Mode: BEHIND_LIVE_EDGE
      Value: 00:00:00
    # Add personalization parameters
    PersonalizationThresholdSeconds: 2
    # Add bumper ads
    Bumper:
      StartUrl: https://example.com/bumper-start.mp4
      EndUrl: https://example.com/bumper-end.mp4
    # Other existing properties...
```

For more information about MediaTailor configuration options, see [Using AWS Elemental MediaTailor to insert ads](#).

Add security features

For broadcast workflows with specific security requirements (like geo-restrictions or protection against DDoS attacks), you can add AWS WAF integration and geo-restrictions:

```
# Create a AWS WAF Web ACL
WebACL:
  Type: AWS::WAFv2::WebACL
```

```
Properties:
  Name: !Sub '${AWS::StackName}-WebACL '
  Scope: CloudFront
  DefaultAction:
    Allow: {}
  VisibilityConfig:
    SampledRequestsEnabled: true
    CloudWatchMetricsEnabled: true
    MetricName: !Sub '${AWS::StackName}-WebACL '
  Rules:
    - Name: RateLimitRule
      Priority: 0
      Action:
        Block: {}
      VisibilityConfig:
        SampledRequestsEnabled: true
        CloudWatchMetricsEnabled: true
        MetricName: RateLimitRule
      Statement:
        RateBasedStatement:
          Limit: 1000
          AggregateKeyType: IP

# Reference the AWS WAF Web ACL in the CloudFront distribution
CloudFrontDistribution:
  Type: AWS::CloudFront::Distribution
  Properties:
    DistributionConfig:
      WebACLId: !GetAtt WebACL.Arn
      # Add geo-restriction
      Restrictions:
        GeoRestriction:
          RestrictionType: whitelist
          Locations:
            - US
            - CA
            - GB
      # Other existing properties...
```

For more information about CloudFormation templates, see the [AWS CloudFormation User Guide](#).

For broadcast-specific CloudFormation templates and examples, see the [AWS Media Services Tools GitHub repository](#).

AWS CloudFormation template reference for AWS Elemental MediaTailor and Amazon CloudFront integration

AWS Elemental MediaTailor integration with Amazon CloudFront can be automated using the following complete AWS CloudFormation template:

```
AWSTemplateFormatVersion: '2010-09-09'
Description: |
  CloudFormation template that sets up AWS Elemental MediaTailor integration with
  CloudFront Distribution
  for server-side ad insertion. This template supports various content origins
  including MediaPackage, Amazon S3,
  and custom origins, making it versatile for different streaming architectures.

Parameters:
  AdServerUrl:
    Type: String
    Default: 'https://d1kbmkziz9rksx.CloudFront.net/VASTEndpoint.xml'
    Description: URL of the VAST ad server for dynamic ad insertion. Static VAST
    endpoint provided for testing.

  ContentOriginDomainName:
    Type: String
    Description: |
      Domain name of your content origin without protocol (e.g., mediapackage-
domain.mediapackagev2.us-west-2.amazonaws.com,
      mybucket.s3.amazonaws.com, or custom-origin.example.com).
      Do not include http:// or https:// prefixes or any paths.
      AllowedPattern: "^(([a-zA-Z0-9]|[a-zA-Z0-9][a-zA-Z0-9\\-]*[a-zA-Z0-9])\\.)*([A-Za-
      z0-9]|[A-Za-z0-9][A-Za-z0-9\\-]*[A-Za-z0-9])$"
      ConstraintDescription: Must be a valid domain name (e.g., example.com) without
      protocol or path components. IP addresses are not allowed.

  ContentOriginType:
    Type: String
    AllowedValues:
      - mediapackagev2
      - s3
      - custom
    Default: mediapackagev2
    Description: |
      The type of content origin:
      - mediapackagev2: AWS Elemental MediaPackage V2
```

- s3: Amazon S3 bucket
- custom: Any other custom origin

Resources:

```
#-----
# Origin Access Control (for securing MediaPackage V2 and Amazon S3 origins)
#-----
CloudFrontOriginAccessControl:
  Type: AWS::CloudFront::OriginAccessControl
  Condition: IsNotCustomOrigin
  Properties:
    OriginAccessControlConfig:
      Name: !Sub '${AWS::StackName}-OAC'
      OriginAccessControlOriginType: !Ref ContentOriginType
      SigningBehavior: always
      SigningProtocol: sigv4
      Description: Origin Access Control for content origin

#-----
# MediaTailor Playback Configuration
#-----
MediaTailorPlaybackConfig:
  Type: AWS::MediaTailor::PlaybackConfiguration
  Properties:
    Name: !Sub '${AWS::StackName}-PlaybackConfig'
    # The video content source should point to your CloudFront distribution
    VideoContentSourceUrl: !Sub 'https://${CloudFrontDistribution.DomainName}/'
    # The Ad Decision Server URL is where MediaTailor will request ads
    AdDecisionServerUrl: !Ref AdServerUrl
    # Configuration for pre-roll ads during live streams
    LivePreRollConfiguration:
      AdDecisionServerUrl: !Ref AdServerUrl
      MaxDurationSeconds: 30
    # CDN configuration for integrating with CloudFront
    CdnConfiguration:
      AdSegmentUrlPrefix: '/'
      ContentSegmentUrlPrefix: '/'
    # Set a reasonable manifest segment timeout
    ManifestProcessingRules:
      AdMarkerPassthrough:
        Enabled: false

#-----
# CloudFront Distribution
```

```

#-----
CloudFrontDistribution:
  Type: AWS::CloudFront::Distribution
  Properties:
    DistributionConfig:
      Enabled: true
      HttpVersion: http2and3
      IPV6Enabled: true
      Comment: !Sub 'Distribution for MediaTailor ad insertion with
${ContentOriginType} origin'

      # Default cache behavior points to the content origin
      DefaultCacheBehavior:
        TargetOriginId: ContentOrigin
        ViewerProtocolPolicy: 'https-only'
        # Using managed policies for optimal performance and simplicity
        CachePolicyId: 658327ea-f89d-4fab-a63d-7e88639e58f6 # Managed-
CachingOptimized
        OriginRequestPolicyId: 88a5eaf4-2fd4-4709-b370-b4c650ea3fcf # Managed-
HostHeaderOnly
        ResponseHeadersPolicyId: eaab4381-ed33-4a86-88ca-d9558dc6cd63 # Managed-
CORS-with-preflight-and-SecurityHeadersPolicy
        Compress: true

      # Define all the origins needed for the workflow
      Origins:
        # Main content origin (MediaPackage, Amazon S3, or Custom)
        - Id: ContentOrigin
          DomainName: !Ref ContentOriginDomainName
          # Apply Origin Access Control for secure origins
          OriginAccessControlId: !If [IsNotCustomOrigin, !GetAtt
CloudFrontOriginAccessControl.Id, !Ref "AWS::NoValue"]
          # For custom origins, we need a CustomOriginConfig
          CustomOriginConfig:
            OriginProtocolPolicy: 'https-only'
            OriginSSLProtocols:
              - TLSv1.2
            OriginKeepaliveTimeout: 5
            OriginReadTimeout: 30
            HTTPPort: 80
            HTTPSPort: 443

        # MediaTailor Manifests Origin - handles manifest manipulation for ad
insertion

```

```

- Id: MediaTailorManifests
DomainName: !Sub 'manifests.mediatailor.${AWS::Region}.amazonaws.com'
CustomOriginConfig:
  OriginProtocolPolicy: 'https-only'
  OriginSSLProtocols:
    - TLSv1.2
  OriginKeepaliveTimeout: 5
  OriginReadTimeout: 30
# Origin Shield improves caching efficiency
OriginShield:
  Enabled: true
  OriginShieldRegion: !Ref AWS::Region

# MediaTailor Segments Origin - handles personalized ads
- Id: MediaTailorSegments
DomainName: !Sub 'segments.mediatailor.${AWS::Region}.amazonaws.com'
CustomOriginConfig:
  OriginProtocolPolicy: 'https-only'
  OriginSSLProtocols:
    - TLSv1.2
  OriginKeepaliveTimeout: 5
  OriginReadTimeout: 30

# Cache behaviors to route specific request patterns to the right origin
CacheBehaviors:
  # Handle MediaTailor segment requests for ad content which are cache-able
  - PathPattern: '/tm/*'
    TargetOriginId: MediaTailorSegments
    ViewerProtocolPolicy: 'https-only'
    CachePolicyId: 658327ea-f89d-4fab-a63d-7e88639e58f6 # Managed-
CachingOptimized
    OriginRequestPolicyId: 88a5eaf4-2fd4-4709-b370-b4c650ea3fcf # Managed-
HostHeaderOnly
    ResponseHeadersPolicyId: eaab4381-ed33-4a86-88ca-d9558dc6cd63 # Managed-
CORS-with-preflight-and-SecurityHeadersPolicy
    Compress: true

  # Handle MediaTailor interstitial (SGAI) media requests which are cache-able
  - PathPattern: '/v1/i-media/*'
    TargetOriginId: MediaTailorManifests
    ViewerProtocolPolicy: 'https-only'
    CachePolicyId: 658327ea-f89d-4fab-a63d-7e88639e58f6 # Managed-
CachingOptimized

```

```

        OriginRequestPolicyId: 88a5eaf4-2fd4-4709-b370-b4c650ea3fcf # Managed-
HostHeaderOnly
        ResponseHeadersPolicyId: eaab4381-ed33-4a86-88ca-d9558dc6cd63 # Managed-
CORS-with-preflight-and-SecurityHeadersPolicy
        Compress: true

        # Handle MediaTailor Personalized manifests which are not cache-able
        - PathPattern: '/v1/*'
          TargetOriginId: MediaTailorManifests
          ViewerProtocolPolicy: 'https-only'
          CachePolicyId: 4135ea2d-6df8-44a3-9df3-4b5a84be39ad # Managed-
CachingDisabled
        OriginRequestPolicyId: 59781a5b-3903-41f3-afcb-af62929ccde1 # Managed-
AllViewer
        ResponseHeadersPolicyId: eaab4381-ed33-4a86-88ca-d9558dc6cd63 # Managed-
CORS-with-preflight-and-SecurityHeadersPolicy
        Compress: true

        # Handle MediaTailor segment *redirect* requests which are not cache-able
        (used for server side reporting)
        - PathPattern: '/segment/*'
          TargetOriginId: MediaTailorManifests
          ViewerProtocolPolicy: 'https-only'
          CachePolicyId: 4135ea2d-6df8-44a3-9df3-4b5a84be39ad # Managed-
CachingDisabled
        OriginRequestPolicyId: 59781a5b-3903-41f3-afcb-af62929ccde1 # Managed-
AllViewer
        ResponseHeadersPolicyId: eaab4381-ed33-4a86-88ca-d9558dc6cd63 # Managed-
CORS-with-preflight-and-SecurityHeadersPolicy
        Compress: true

Conditions:
  IsNotCustomOrigin: !Not [!Equals [!Ref ContentOriginType, 'custom']]

Outputs:
  CloudFrontDomainName:
    Description: Domain name of the CloudFront distribution
    Value: !GetAtt CloudFrontDistribution.DomainName

  HlsManifestUrl:
    Description: URL for HLS manifest with ads inserted (append your manifest path)
    Value: !Sub 'https://
${CloudFrontDistribution.DomainName}${MediaTailorPlaybackConfig.HlsConfiguration.ManifestEndpoint

```



```
DashManifestUrl:
  Description: URL for DASH manifest with ads inserted (append your manifest path)
  Value: !Sub 'https://
${CloudFrontDistribution.DomainName}${MediaTailorPlaybackConfig.DashConfiguration.ManifestEndpointPath}

MediaTailorPlaybackConfigName:
  Description: Name of the MediaTailor playback configuration
  Value: !Ref MediaTailorPlaybackConfig
```

Production-ready CloudFront configuration for MediaTailor

This CloudFront distribution configuration provides everything you need to deliver MediaTailor content with server-side ad insertion at scale. Copy this configuration and customize it with your specific origins and requirements.

What this configuration accomplishes

This configuration creates a production-ready CloudFront distribution that handles all MediaTailor request types with optimal caching and performance. It includes three origins (your content, MediaTailor segments, and MediaTailor manifests) with four cache behaviors that route requests correctly and cache content appropriately.

When to use this configuration

This setup is ideal for live streaming, video-on-demand, and hybrid workflows that require server-side ad insertion.

Three-origin architecture

MediaTailor uses a three-origin architecture pattern to optimize content delivery and ad insertion performance. Each origin serves a specific purpose in the ad insertion workflow:

Your content origin

This is your true content origin that feeds MediaTailor. For example, this could be AWS Elemental MediaPackage V2 or another content delivery service. This origin serves your original content before ad insertion. Common examples include:

- MediaPackage V2 packaging configurations
- Third-party content delivery networks

- On-premises streaming servers
- Amazon S3 buckets with static content

MediaTailor segments origin

This origin uses the hostname `segments.mediatailor.region.amazonaws.com` and serves the actual ad segments after MediaTailor has encoded them. These are the video segments that contain the advertisements. This origin handles:

- Transcoded ad segments in the same format as your content
- Redirected requests from the `/segment/*` path pattern
- Ad segments that have been processed for server-side ad insertion

MediaTailor manifests origin

This origin uses the hostname `manifests.mediatailor.region.amazonaws.com` and can be used as a regional hostname for playback configurations in the specified AWS Region. MediaTailor selects the correct playback configuration based on the path in the request. This origin provides:

- Personalized HLS and DASH manifests with viewer-specific ad insertion
- Server-guided ad insertion (SGAI) manifests for cacheable content
- Ad tracking and beacon handling for server-side reporting

With the origin hostname `manifests.mediatailor.region.amazonaws.com`, you can have multiple playback configurations that work with the same CloudFront distribution if they're in the specified Region and you include the playback configuration name in the request path. For example:

- `https://your-distribution.cloudfront.net/v1/master/playback-config-1/manifest.m3u8`
- `https://your-distribution.cloudfront.net/v1/master/playback-config-2/manifest.m3u8`

Select origin request policies based on content type to prevent cache poisoning while ensuring proper functionality. The key distinction is between cacheable and non-cacheable content:

- **Manifests (non-cacheable):** Use `AllViewer` to forward all headers needed for dynamic content. Since manifests aren't cached, there's no cache poisoning risk.

- **Segments (cacheable):** Use None for optimal performance.
- **S3 origins:** Use CORS-S3Origin for Amazon S3 buckets
- **MediaPackage origins:** Use CORS-S3Origin for MediaPackage V2 endpoints

Origins							Edit	Delete	Create origin
Filter origins by property or value							< 1 > ⚙		
	Origin name	Origin domain	Origin path	Origin type	Origin Shield region	Origin access			
☐	ContentOrigin	zrljic.egress.kv9ydh.mediapackagev2.us-west-2.amazonaws.com		MediaPackage V2	us-west-2	E2SRJY3NEMC428			
☐	MediaTailorSegments	segments.mediatailor.us-west-2.amazonaws.com		MediaTailor	-	-			
☐	MediaTailorManifests	manifests.mediatailor.us-west-2.amazonaws.com		MediaTailor	us-west-2	-			

Cache behavior precedence and configuration

MediaTailor requires specific cache behavior configurations to handle different types of requests properly. The precedence of cache behaviors is critical because CDNs process them in order (smallest to largest) and use the behavior for the first matching path pattern. Understanding this precedence is essential for troubleshooting:

- **Precedence 0:** Most specific patterns (like `/tm/*`) are evaluated first
- **Higher precedence numbers:** Less specific patterns are evaluated in order
- **Default behavior:** Catches all requests that don't match other patterns

If requests aren't behaving as expected, check that your path patterns don't overlap in unintended ways.

General Security Origins Behaviors Error pages Invalidations Tags Logging							
Behaviors							
Filter behaviors by property or value							
	Preced...	Path patt...	Origin or origin group	Viewer pr...	Cache policy name	Origin request policy name	Response headers policy name
☐	0	/tm/*	MediaTailorSegments	HTTPS only	Managed-CachingOptimized	Managed-CORS-S3Origin	Managed-CORS-with-preflight-and-SecurityHeadersPolicy
☐	1	/i-media/*	MediaTailorManifests	HTTPS only	Managed-CachingOptimized	Managed-CORS-S3Origin	Managed-CORS-with-preflight-and-SecurityHeadersPolicy
☐	2	/v1/*	MediaTailorManifests	HTTPS only	Managed-CachingDisabled	Managed-AllViewerAndCloudFrontHeaders-2022-06	Managed-CORS-with-preflight-and-SecurityHeadersPolicy
☐	3	/segment/*	MediaTailorManifests	HTTPS only	Managed-CachingDisabled	Managed-AllViewerAndCloudFrontHeaders-2022-06	Managed-CORS-with-preflight-and-SecurityHeadersPolicy
☐	4	Default (*)	ContentOrigin	HTTPS only	Managed-CachingOptimized	CustomOriginRequestPolicyWithWithLLHLSDirectives	Managed-CORS-with-preflight-and-SecurityHeadersPolicy

Precedence 0: Ad segments path behavior

This behavior handles the redirected requests from the segment path behavior, serving the actual ad segments. CloudFront applies the following behaviors to all requests with a `/tm/*` path pattern.

This is the highest priority behavior because ad segment delivery is critical for uninterrupted playback.

- **Path pattern:** `/tm/*`

Example URLs that match this pattern:

- `https://your-distribution.cloudfront.net/tm/ad-segment-001.ts`
- `https://your-distribution.cloudfront.net/tm/transcoded-ad.m4s`
- **Origin:** The origin that you created with the `segments.mediatailor.region.amazonaws.com` domain.

This is **MediaTailorSegments** in the example in the preceding section on origins.

- **Cache policy:** `Managed-CachingOptimized`

For details about what's included in the CloudFront managed cache policy, see [CachingOptimized](#) in the CloudFront user guide. You can also use these same settings from the managed policy in your third-party CDN.

- **Origin request policy:** `None`
- **Response headers policy:** `Managed-CORS-with-preflight-and-SecurityHeadersPolicy`

For details about what's included in the response headers policy, see [CORS-with-preflight-and-SecurityHeadersPolicy](#) in the CloudFront user guide.

Adapting these settings to other CDNs

If you're using a CDN other than CloudFront, look for equivalent settings that accomplish the following.

Path pattern matching

Configure a specific behavior for the `/tm/*` path pattern to handle MediaTailor ad segments

Cache key configuration

Include the `Origin` header in your cache key to ensure responses are cached separately for different origins

Header forwarding

Forward the `Origin` header and other CORS-related headers to the origin

Response header management

Configure your CDN to ensure the `Access-Control-Allow-Origin` header is present in responses

The specific terminology and configuration options will vary by CDN provider, but the underlying principles remain the same.

Precedence 1: Server-guided ad insertion behavior

This behavior handles [MediaTailor server-guided ad insertion overview and implementation](#) (SGAI) requests when customers configure guided mode, which provides cacheable manifests. CloudFront applies the following behaviors to all requests with a `/v1/i-media/*` path pattern. SGAi allows for better caching performance because the manifests are not viewer-specific.

- **Path pattern:** `/v1/i-media/*` (iMedia path for SGAi)

Example URLs that match this pattern:

- `https://your-distribution.cloudfront.net/v1/i-media/your-config/manifest.m3u8`
- `https://your-distribution.cloudfront.net/v1/i-media/your-config/playlist.mpd`
- **Origin:** The origin that you created with the `manifests.mediatailor.region.amazonaws.com` domain.

This is **MediaTailorManifests** in the example in the preceding section on origins.

- **Cache policy:** `Managed-CachingOptimized`

For details about what's included in the CloudFront managed cache policy, see [CachingOptimized](#) in the CloudFront user guide. You can also use these same settings from the managed policy in your third-party CDN.

- **Origin request policy:** `None`
- **Response headers policy:** `Managed-CORS-with-preflight-and-SecurityHeadersPolicy`

For details about what's included in the response headers policy, see [CORS-with-preflight-and-SecurityHeadersPolicy](#) in the CloudFront user guide.

Precedence 2: Personalized manifest behavior

This behavior handles personalized manifest requests. CloudFront applies the following behaviors to all requests with a `/v1/*` path pattern. CloudFront applies the following behaviors and doesn't cache personalized manifests because they contain viewer-specific ad content URLs. These behaviors apply to all requests that have a `/v1/*` path pattern. This is the core MediaTailor functionality where each viewer receives a unique manifest with personalized ad insertion.

- **Path pattern:** `/v1/*` (standard V1 MediaTailor requests)

Example URLs that match this pattern:

- `https://your-distribution.cloudfront.net/v1/master/your-config/manifest.m3u8`
- `https://your-distribution.cloudfront.net/v1/dash/your-config/manifest.mpd`
- **Origin:** The origin that you created with the `manifests.mediatailor.region.amazonaws.com` domain.

This is **MediaTailorManifests** in the example in the preceding section on origins.

- **Cache policy:** `Managed-CachingDisabled`

For details about what's included in the cache policy, see [CachingDisabled](#) in the CloudFront user guide.

- **Origin request policy:** `AllViewer`

For personalized manifests, use the `AllViewer` policy to forward all headers needed for dynamic content.

- **Response headers policy:** `Managed-CORS-with-preflight-and-SecurityHeadersPolicy`

For details about what's included in the response headers policy, see [CORS-with-preflight-and-SecurityHeadersPolicy](#) in the CloudFront user guide.

Accept-Encoding header

We recommend that your CDN preserves the `Accept-Encoding` header from viewers. This header gives MediaTailor instruction on compressing personalized manifests.

In CloudFront, the `AllViewerAndCloudFrontHeaders` origin request policy includes passthrough of the `Accept-Encoding` header from the viewer. If you use a different CDN, ensure that it preserves this header.

The following is how MediaTailor handles the `Accept-Encoding` header.

- **Legacy devices:** Older smart TVs that don't support gzip won't send the `Accept-Encoding` header, so MediaTailor returns uncompressed manifests
- **Modern devices:** iPhones, Chrome browsers, and other modern clients send the `Accept-Encoding` header, allowing MediaTailor to compress manifests before delivery

Precedence 3: Server-side beacon path behavior

This behavior handles requests to MediaTailor that result in redirects for [Server-side tracking](#). These requests are essential for tracking beacons, so every request must be processed by MediaTailor. CloudFront applies the following behaviors to all requests with a `/segment/*` path pattern.

- **Path pattern:** `/segment/*`

Example URLs that match this pattern:

- `https://your-distribution.cloudfront.net/segment/tracking-beacon-123`
- `https://your-distribution.cloudfront.net/segment/ad-request-456.ts`
- **Origin:** The origin that you created with the `manifests.mediatailor.region.amazonaws.com` domain.

This is **MediaTailorManifests** in the example in the preceding section on origins.

- **Cache policy:** `Managed-CachingDisabled`

For details about what's included in the cache policy, see [CachingDisabled](#) in the CloudFront user guide.

- **Origin request policy:** `AllViewer`

For server-side beacon requests, use the `AllViewer` policy to forward all headers needed for tracking. Since these requests aren't cached, there's no cache poisoning risk.

- **Response headers policy:** `Managed-CORS-with-preflight-and-SecurityHeadersPolicy`

For details about what's included in the response headers policy, see [CORS-with-preflight-and-SecurityHeadersPolicy](#) in the CloudFront user guide.

When MediaTailor processes these requests, it returns a 302 redirect response with a path that points to the actual segment location. For example, a request to `/segment/ad123.ts` might redirect to `/tm/encoded-ad-segment.ts` on the segments origin.

Precedence 4: Content origin path behavior

If the request path doesn't match any of the other patterns, CloudFront applies the default behavior. This behavior sends requests directly to the content origin, with no processing from MediaTailor. This allows direct access to your content origin (such as MediaPackage V2) when needed. CloudFront applies the following behaviors to all requests that don't include any of the prior path patterns.

- **Path pattern:** (*)
- **Origin:** The origin that you created with the domain for your content origin.

This is **ContentOrigin** in the example in the preceding section on origins.

- **Cache policy:** Managed-CachingOptimized

For details about what's included in the CloudFront managed cache policy, see [CachingOptimized](#) in the CloudFront user guide. You can also use these same settings from the managed policy in your third-party CDN.

Note

For low latency HLS implementations, consider using a custom caching policy with Low Latency HLS (LLH) directives instead of the standard CachingOptimized policy.

- **Origin request policy:** None
- **Response headers policy:** Managed-CORS-with-preflight-and-SecurityHeadersPolicy

Although the default content origin behavior doesn't typically face the same CORS cache poisoning risks as the ad segment behavior, it's still recommended to use the Managed-CORS-with-preflight-and-SecurityHeadersPolicy response headers policy and include the

Origin header in your cache key. This ensures consistent CORS handling across all content types and prevents potential playback issues in web-based players.

For content segments, the Managed-CachingOptimized cache policy provides good performance while the Managed-CORS-with-preflight-and-SecurityHeadersPolicy response headers policy ensures proper CORS handling. This combination allows for efficient caching while maintaining compatibility with web-based players that require CORS headers.

Applying consistent CORS handling across both ad segments and content segments creates a more reliable playback experience and simplifies troubleshooting. Without proper CORS configuration, players might experience inconsistent behavior when transitioning between content and ads.

For details about what's included in the response headers policy, see [CORS-with-preflight-and-SecurityHeadersPolicy](#) in the CloudFront user guide.

Get support and troubleshooting help for CDN and MediaTailor integrations

AWS Elemental MediaTailor CDN integration issues can be complex to diagnose and resolve. Use this guide when you need additional help with CDN and MediaTailor integration issues that you can't resolve through self-service troubleshooting.

Before escalating, try these self-service options:

- Follow the troubleshooting steps in [Troubleshoot CDN integration](#)
- Analyze your logs and error codes using [CDN integration log analysis reference](#)
- Review your monitoring setup with [Set up monitoring tools](#)
- Check the [MediaTailor troubleshooting guide](#) for service-specific issues
- Search [AWS re:Post](#) for similar issues and community solutions
- Review [AWS Knowledge Center](#) for common integration patterns

Topics

- [When to escalate MediaTailor CDN issues to AWS Support](#)
- [Gather MediaTailor CDN information before contacting support](#)

- [MediaTailor CDN integration support resources and channels](#)
- [MediaTailor CDN integration support case best practices](#)

When to escalate MediaTailor CDN issues to AWS Support

AWS Elemental MediaTailor CDN integration issues should be escalated to AWS Support when self-service troubleshooting doesn't resolve the problem. Consider escalating to AWS Support when:

Note: Technical support requires a paid AWS Support plan. For support plan details and response times, see [AWS Support plans](#).

- Issues affect production traffic or revenue
- You've followed all troubleshooting steps without resolution
- Error patterns suggest service-level issues
- You need assistance with complex configuration scenarios
- Performance issues persist despite optimization efforts

Gather MediaTailor CDN information before contacting support

AWS Elemental MediaTailor CDN integration support cases require specific configuration and diagnostic information for effective troubleshooting. Before contacting AWS Support, gather this information to help expedite resolution:

Tip: Having this information ready before creating your support case will significantly reduce resolution time and help support engineers understand your specific configuration.

- **MediaTailor configuration details:**
 - Configuration name and AWS Region
 - Playback configuration ARN
 - ADS URL and integration type
- **CDN configuration details:**
 - CDN distribution ID or configuration name
 - Cache behavior configurations for manifests and segments
 - Origin configuration and routing rules
- **Error information:**

- Specific error messages and HTTP status codes
- Timestamps when issues occur
- Sample URLs that demonstrate the problem
- CDN and MediaTailor log entries related to the issue
- **Testing information:**
 - Steps you've already taken to troubleshoot
 - Devices and players where the issue occurs
 - Whether the issue affects all content or specific streams
 - Frequency and pattern of the issue (intermittent, consistent, time-based)

MediaTailor CDN integration support resources and channels

AWS Elemental MediaTailor CDN integration support is available through multiple channels to help you resolve issues and optimize your implementation:

- **AWS Support:** Create a support case through the AWS Management Console for technical assistance

Access: [AWS Support Center](#)

- **AWS re:Post:** Community-driven Q&A platform for AWS-related questions and community support

Access: [AWS re:Post](#)

- **AWS Documentation:** Comprehensive guides for MediaTailor and CDN services

Access: [MediaTailor Documentation](#) and [CloudFront Documentation](#)

- **AWS Training:** Courses and certifications for media services and CDN optimization

Access: [AWS Training and Certification](#)

- **AWS Knowledge Center:** Curated articles for common AWS issues and best practices

Access: [AWS Knowledge Center](#)

- **AWS Trusted Advisor:** Automated recommendations for optimization and best practices

Access: [AWS Trusted Advisor](#)

- **AWS Personal Health Dashboard:** Service health and maintenance notifications

Access: [AWS Personal Health Dashboard](#)

MediaTailor CDN integration support case best practices

AWS Elemental MediaTailor CDN integration support cases are resolved more efficiently when you follow these best practices. To get the fastest resolution:

- Choose the appropriate severity level based on business impact
- Provide all relevant information in your initial case submission
- Include specific examples and reproduction steps
- Attach relevant log files and configuration screenshots
- Respond promptly to support engineer requests for additional information

Additional support resources:

- [AWS Support case management](#) - Guide to creating and managing support cases
- [AWS Support plans](#) - Compare support plan features and response times
- [Prepare to support workloads](#) - Well-Architected guidance for operational readiness
- [AWS Well-Architected Framework](#) - Best practices for building and operating workloads on AWS

Security in AWS Elemental MediaTailor

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from data centers and network architectures that are built to meet the requirements of the most security-sensitive organizations.

Security is a shared responsibility between AWS and you. The [shared responsibility model](#) describes this as security *of* the cloud and security *in* the cloud:

- **Security of the cloud** – AWS is responsible for protecting the infrastructure that runs AWS services in the AWS Cloud. AWS also provides you with services that you can use securely. Third-party auditors regularly test and verify the effectiveness of our security as part of the [AWS Compliance Programs](#). To learn about the compliance programs that apply to AWS Elemental MediaTailor, see [AWS Services in Scope by Compliance Program](#).
- **Security in the cloud** – Your responsibility is determined by the AWS service that you use. You are also responsible for other factors including the sensitivity of your data, your company's requirements, and applicable laws and regulations.

This documentation helps you understand how to apply the shared responsibility model when using MediaTailor. The following topics show you how to configure MediaTailor to meet your security and compliance objectives. You also learn how to use other AWS services that help you to monitor and secure your MediaTailor resources.

Topics

- [Data protection in AWS Elemental MediaTailor](#)
- [Identity and Access Management for AWS Elemental MediaTailor](#)
- [Compliance validation for AWS Elemental MediaTailor](#)
- [Resilience in AWS Elemental MediaTailor](#)
- [Infrastructure security in MediaTailor](#)
- [Cross-service confused deputy prevention](#)
- [Logging and monitoring in MediaTailor](#)

Data protection in AWS Elemental MediaTailor

The AWS [shared responsibility model](#) applies to data protection in AWS Elemental MediaTailor. As described in this model, AWS is responsible for protecting the global infrastructure that runs all of the AWS Cloud. You are responsible for maintaining control over your content that is hosted on this infrastructure. You are also responsible for the security configuration and management tasks for the AWS services that you use. For more information about data privacy, see [Data Privacy FAQ](#). For information about data protection in Europe, see the [General Data Protection Regulation \(GDPR\) Center](#).

For data protection purposes, we recommend that you protect AWS account credentials and set up individual users with AWS IAM Identity Center or AWS Identity and Access Management (IAM). That way, each user is given only the permissions necessary to fulfill their job duties. We also recommend that you secure your data in the following ways:

- Use multi-factor authentication (MFA) with each account.
- Use SSL/TLS to communicate with AWS resources. We require TLS 1.2 and recommend TLS 1.3.
- Set up API and user activity logging with AWS CloudTrail. For information about using CloudTrail trails to capture AWS activities, see [Working with CloudTrail trails](#) in the *AWS CloudTrail User Guide*.
- Use AWS encryption solutions, along with all default security controls within AWS services.
- Use advanced managed security services such as Amazon Macie, which assists in discovering and securing sensitive data that is stored in Amazon S3.
- If you require FIPS 140-3 validated cryptographic modules when accessing AWS through a command line interface or an API, use a FIPS endpoint. For more information about the available FIPS endpoints, see [Federal Information Processing Standard \(FIPS\) 140-3](#).

We strongly recommend that you never put confidential or sensitive information, such as your customers' email addresses, into tags or free-form text fields such as a **Name** field. This includes when you work with MediaTailor or other AWS services using the console, API, AWS CLI, or AWS SDKs. Any data that you enter into tags or free-form text fields used for names may be used for billing or diagnostic logs. If you provide a URL to an external server, we strongly recommend that you do not include credentials information in the URL to validate your request to that server.

Data encryption

AWS Elemental MediaTailor doesn't encrypt or decrypt data in its management of content manifests or in its communication with servers, CDNs, or players. MediaTailor doesn't require that you supply any customer data or other sensitive information.

Don't put sensitive information, like customer account numbers, credit card information, or sign-in credentials, into free-form fields or query parameters. This applies to all use of AWS Elemental MediaTailor, including the console, API, SDKs, and the AWS Command Line Interface (AWS CLI). Any data that you enter into the service might get picked up for inclusion in diagnostic logs.

When you provide a URL to an external server, don't include unencrypted credentials information in the URL to validate your request to that server.

Identity and Access Management for AWS Elemental MediaTailor

AWS Identity and Access Management (IAM) is an AWS service that helps an administrator securely control access to AWS resources. IAM administrators control who can be *authenticated* (signed in) and *authorized* (have permissions) to use MediaTailor resources. IAM is an AWS service that you can use with no additional charge.

Topics

- [Audience](#)
- [Authenticating with identities](#)
- [Managing access using policies](#)
- [How AWS Elemental MediaTailor works with IAM](#)
- [Identity-based policy examples for AWS Elemental MediaTailor](#)
- [Resource-based policy examples for AWS Elemental MediaTailor](#)
- [AWS managed policies for AWS Elemental MediaTailor](#)
- [Using service-linked roles for MediaTailor](#)
- [Troubleshooting AWS Elemental MediaTailor identity and access](#)

Audience

How you use AWS Identity and Access Management (IAM) differs based on your role:

- **Service user** - request permissions from your administrator if you cannot access features (see [Troubleshooting AWS Elemental MediaTailor identity and access](#))
- **Service administrator** - determine user access and submit permission requests (see [How AWS Elemental MediaTailor works with IAM](#))
- **IAM administrator** - write policies to manage access (see [Identity-based policy examples for AWS Elemental MediaTailor](#))

Authenticating with identities

Authentication is how you sign in to AWS using your identity credentials. You must be authenticated as the AWS account root user, an IAM user, or by assuming an IAM role.

You can sign in as a federated identity using credentials from an identity source like AWS IAM Identity Center (IAM Identity Center), single sign-on authentication, or Google/Facebook credentials. For more information about signing in, see [How to sign in to your AWS account](#) in the *AWS Sign-In User Guide*.

For programmatic access, AWS provides an SDK and CLI to cryptographically sign requests. For more information, see [AWS Signature Version 4 for API requests](#) in the *IAM User Guide*.

AWS account root user

When you create an AWS account, you begin with one sign-in identity called the AWS account *root user* that has complete access to all AWS services and resources. We strongly recommend that you don't use the root user for everyday tasks. For tasks that require root user credentials, see [Tasks that require root user credentials](#) in the *IAM User Guide*.

Federated identity

As a best practice, require human users to use federation with an identity provider to access AWS services using temporary credentials.

A *federated identity* is a user from your enterprise directory, web identity provider, or Directory Service that accesses AWS services using credentials from an identity source. Federated identities assume roles that provide temporary credentials.

For centralized access management, we recommend AWS IAM Identity Center. For more information, see [What is IAM Identity Center?](#) in the *AWS IAM Identity Center User Guide*.

IAM users and groups

An [IAM user](#) is an identity with specific permissions for a single person or application. We recommend using temporary credentials instead of IAM users with long-term credentials. For more information, see [Require human users to use federation with an identity provider to access AWS using temporary credentials](#) in the *IAM User Guide*.

An [IAM group](#) specifies a collection of IAM users and makes permissions easier to manage for large sets of users. For more information, see [Use cases for IAM users](#) in the *IAM User Guide*.

IAM roles

An [IAM role](#) is an identity with specific permissions that provides temporary credentials. You can assume a role by [switching from a user to an IAM role \(console\)](#) or by calling an AWS CLI or AWS API operation. For more information, see [Methods to assume a role](#) in the *IAM User Guide*.

IAM roles are useful for federated user access, temporary IAM user permissions, cross-account access, cross-service access, and applications running on Amazon EC2. For more information, see [Cross account resource access in IAM](#) in the *IAM User Guide*.

Managing access using policies

You control access in AWS by creating policies and attaching them to AWS identities or resources. A policy defines permissions when associated with an identity or resource. AWS evaluates these policies when a principal makes a request. Most policies are stored in AWS as JSON documents. For more information about JSON policy documents, see [Overview of JSON policies](#) in the *IAM User Guide*.

Using policies, administrators specify who has access to what by defining which **principal** can perform **actions** on what **resources**, and under what **conditions**.

By default, users and roles have no permissions. An IAM administrator creates IAM policies and adds them to roles, which users can then assume. IAM policies define permissions regardless of the method used to perform the operation.

Identity-based policies

Identity-based policies are JSON permissions policy documents that you attach to an identity (user, group, or role). These policies control what actions identities can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see [Define custom IAM permissions with customer managed policies](#) in the *IAM User Guide*.

Identity-based policies can be *inline policies* (embedded directly into a single identity) or *managed policies* (standalone policies attached to multiple identities). To learn how to choose between managed and inline policies, see [Choose between managed policies and inline policies](#) in the *IAM User Guide*.

Resource-based policies

Resource-based policies are JSON policy documents that you attach to a resource. Examples include IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. You must [specify a principal](#) in a resource-based policy.

Resource-based policies are inline policies that are located in that service. You can't use AWS managed policies from IAM in a resource-based policy.

Other policy types

AWS supports additional policy types that can set the maximum permissions granted by more common policy types:

- **Permissions boundaries** – Set the maximum permissions that an identity-based policy can grant to an IAM entity. For more information, see [Permissions boundaries for IAM entities](#) in the *IAM User Guide*.
- **Service control policies (SCPs)** – Specify the maximum permissions for an organization or organizational unit in AWS Organizations. For more information, see [Service control policies](#) in the *AWS Organizations User Guide*.
- **Resource control policies (RCPs)** – Set the maximum available permissions for resources in your accounts. For more information, see [Resource control policies \(RCPs\)](#) in the *AWS Organizations User Guide*.
- **Session policies** – Advanced policies passed as a parameter when creating a temporary session for a role or federated user. For more information, see [Session policies](#) in the *IAM User Guide*.

Multiple policy types

When multiple types of policies apply to a request, the resulting permissions are more complicated to understand. To learn how AWS determines whether to allow a request when multiple policy types are involved, see [Policy evaluation logic](#) in the *IAM User Guide*.

How AWS Elemental MediaTailor works with IAM

Before you use IAM to manage access to MediaTailor, learn what IAM features are available to use with MediaTailor.

IAM features you can use with AWS Elemental MediaTailor

IAM feature	MediaTailor support
Identity-based policies	Yes
Resource-based policies	Yes
Policy actions	Yes
Policy resources	No
Policy condition keys (service-specific)	Yes
ACLs	No
ABAC (tags in policies)	Partial
Temporary credentials	Yes
Principal permissions	Yes
Service roles	No
Service-linked roles	Yes

To get a high-level view of how MediaTailor and other AWS services work with most IAM features, see [AWS services that work with IAM](#) in the *IAM User Guide*.

Identity-based policies for MediaTailor

Supports identity-based policies: Yes

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see [Define custom IAM permissions with customer managed policies](#) in the *IAM User Guide*.

With IAM identity-based policies, you can specify allowed or denied actions and resources as well as the conditions under which actions are allowed or denied. To learn about all of the elements that you can use in a JSON policy, see [IAM JSON policy elements reference](#) in the *IAM User Guide*.

Identity-based policy examples for MediaTailor

To view examples of MediaTailor identity-based policies, see [Identity-based policy examples for AWS Elemental MediaTailor](#).

Resource-based policies within MediaTailor

Supports resource-based policies: Yes

The MediaTailor service supports only one type of resource-based policy. It's called a *channel policy* because it's attached to a channel. This policy defines which principals can perform actions on the channel.

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must [specify a principal](#) in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

To enable cross-account access, you can specify an entire account or IAM entities in another account as the principal in a resource-based policy. For more information, see [Cross account resource access in IAM](#) in the *IAM User Guide*.

To learn how to attach a resource-based policy to a channel, see [Create a channel using the MediaTailor console](#).

Resource-based policy examples within MediaTailor

To view examples of MediaTailor resource-based policies, see [Resource-based policy examples for AWS Elemental MediaTailor](#).

Policy actions for MediaTailor

Supports policy actions: Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The Action element of a JSON policy describes the actions that you can use to allow or deny access in a policy. Include actions in a policy to grant permissions to perform the associated operation.

To see a list of MediaTailor actions, see [Actions defined by AWS Elemental MediaTailor](#) in the *Service Authorization Reference*.

Policy actions in MediaTailor use the following prefix before the action:

```
mediatailor
```

To specify multiple actions in a single statement, separate them with commas.

```
"Action": [  
    "mediatailor:action1",  
    "mediatailor:action2"  
]
```

To view examples of MediaTailor identity-based policies, see [Identity-based policy examples for AWS Elemental MediaTailor](#).

Policy resources for MediaTailor

Supports policy resources: No

AWS Elemental MediaTailor doesn't support specifying resource ARNs in a policy.

Policy condition keys for MediaTailor

Supports service-specific policy condition keys: Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Condition` element specifies when statements execute based on defined criteria. You can create conditional expressions that use [condition operators](#), such as equals or less than, to match the condition in the policy with values in the request. To see all AWS global condition keys, see [AWS global condition context keys](#) in the *IAM User Guide*.

For a list of MediaTailor condition keys, see [Condition keys for AWS Elemental MediaTailor](#) in the *Service Authorization Reference*. To learn with which actions and resources you can use a condition key, see [Actions defined by AWS Elemental MediaTailor](#).

AWS Elemental MediaTailor doesn't provide service-specific condition keys, but it does support using some global condition keys. To see all AWS global condition keys, see [AWS Global Condition Context Keys](#) in the *AWS Identity and Access Management User Guide*.

ACLs in MediaTailor

Supports ACLs: No

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

ABAC with MediaTailor

Supports ABAC (tags in policies): Partial

Attribute-based access control (ABAC) is an authorization strategy that defines permissions based on attributes called tags. You can attach tags to IAM entities and AWS resources, then design ABAC policies to allow operations when the principal's tag matches the tag on the resource.

To control access based on tags, you provide tag information in the [condition element](#) of a policy using the `aws:ResourceTag/key-name`, `aws:RequestTag/key-name`, or `aws:TagKeys` condition keys.

If a service supports all three condition keys for every resource type, then the value is **Yes** for the service. If a service supports all three condition keys for only some resource types, then the value is **Partial**.

For more information about ABAC, see [Define permissions with ABAC authorization](#) in the *IAM User Guide*. To view a tutorial with steps for setting up ABAC, see [Use attribute-based access control \(ABAC\)](#) in the *IAM User Guide*.

For MediaTailor, use the value **Partial**.

Using temporary credentials with MediaTailor

Supports temporary credentials: Yes

Temporary credentials provide short-term access to AWS resources and are automatically created when you use federation or switch roles. AWS recommends that you dynamically generate temporary credentials instead of using long-term access keys. For more information, see [Temporary security credentials in IAM](#) and [AWS services that work with IAM](#) in the *IAM User Guide*.

Cross-service principal permissions for MediaTailor

Supports forward access sessions (FAS): Yes

Forward access sessions (FAS) use the permissions of the principal calling an AWS service, combined with the requesting AWS service to make requests to downstream services. For policy details when making FAS requests, see [Forward access sessions](#).

Service roles for MediaTailor

Supports service roles: No

AWS Elemental MediaTailor doesn't support service roles.

Service-linked roles for MediaTailor

Supports service-linked roles: Yes

A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your AWS account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.

For details about creating or managing MediaTailor service-linked roles, see [Using service-linked roles for MediaTailor](#).

Identity-based policy examples for AWS Elemental MediaTailor

By default, users and roles don't have permission to create or modify MediaTailor resources. To grant users permission to perform actions on the resources that they need, an IAM administrator can create IAM policies.

To learn how to create an IAM identity-based policy by using these example JSON policy documents, see [Create IAM policies \(console\)](#) in the *IAM User Guide*.

For details about actions and resource types defined by MediaTailor, including the format of the ARNs for each of the resource types, see [Actions, resources, and condition keys for AWS Elemental MediaTailor](#) in the *Service Authorization Reference*.

Topics

- [Policy best practices](#)
- [Using the MediaTailor console](#)
- [Allow users to view their own permissions](#)

Policy best practices

Identity-based policies determine whether someone can create, access, or delete MediaTailor resources in your account. These actions can incur costs for your AWS account. When you create or edit identity-based policies, follow these guidelines and recommendations:

- **Get started with AWS managed policies and move toward least-privilege permissions** – To get started granting permissions to your users and workloads, use the *AWS managed policies* that grant permissions for many common use cases. They are available in your AWS account. We recommend that you reduce permissions further by defining AWS customer managed policies that are specific to your use cases. For more information, see [AWS managed policies](#) or [AWS managed policies for job functions](#) in the *IAM User Guide*.
- **Apply least-privilege permissions** – When you set permissions with IAM policies, grant only the permissions required to perform a task. You do this by defining the actions that can be taken on specific resources under specific conditions, also known as *least-privilege permissions*. For more information about using IAM to apply permissions, see [Policies and permissions in IAM](#) in the *IAM User Guide*.

- **Use conditions in IAM policies to further restrict access** – You can add a condition to your policies to limit access to actions and resources. For example, you can write a policy condition to specify that all requests must be sent using SSL. You can also use conditions to grant access to service actions if they are used through a specific AWS service, such as CloudFormation. For more information, see [IAM JSON policy elements: Condition](#) in the *IAM User Guide*.
- **Use IAM Access Analyzer to validate your IAM policies to ensure secure and functional permissions** – IAM Access Analyzer validates new and existing policies so that the policies adhere to the IAM policy language (JSON) and IAM best practices. IAM Access Analyzer provides more than 100 policy checks and actionable recommendations to help you author secure and functional policies. For more information, see [Validate policies with IAM Access Analyzer](#) in the *IAM User Guide*.
- **Require multi-factor authentication (MFA)** – If you have a scenario that requires IAM users or a root user in your AWS account, turn on MFA for additional security. To require MFA when API operations are called, add MFA conditions to your policies. For more information, see [Secure API access with MFA](#) in the *IAM User Guide*.

For more information about best practices in IAM, see [Security best practices in IAM](#) in the *IAM User Guide*.

Using the MediaTailor console

To access the AWS Elemental MediaTailor console, you must have a minimum set of permissions. These permissions must allow you to list and view details about the MediaTailor resources in your AWS account. If you create an identity-based policy that is more restrictive than the minimum required permissions, the console won't function as intended for entities (users or roles) with that policy.

You don't need to allow minimum console permissions for users that are making calls only to the AWS CLI or the AWS API. Instead, allow access to only the actions that match the API operation that they're trying to perform.

To ensure that users and roles can still use the MediaTailor console, also attach the MediaTailor *ConsoleAccess* or *ReadOnly* AWS managed policy to the entities. For more information, see [Adding permissions to a user](#) in the *IAM User Guide*.

Allow users to view their own permissions

This example shows how you might create a policy that allows IAM users to view the inline and managed policies that are attached to their user identity. This policy includes permissions to complete this action on the console or programmatically using the AWS CLI or AWS API.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ViewOwnUserInfo",
      "Effect": "Allow",
      "Action": [
        "iam:GetUserPolicy",
        "iam:ListGroupsForUser",
        "iam:ListAttachedUserPolicies",
        "iam:ListUserPolicies",
        "iam:GetUser"
      ],
      "Resource": ["arn:aws:iam::*:user/${aws:username}"]
    },
    {
      "Sid": "NavigateInConsole",
      "Effect": "Allow",
      "Action": [
        "iam:GetGroupPolicy",
        "iam:GetPolicyVersion",
        "iam:GetPolicy",
        "iam:ListAttachedGroupPolicies",
        "iam:ListGroupPolicies",
        "iam:ListPolicyVersions",
        "iam:ListPolicies",
        "iam:ListUsers"
      ],
      "Resource": "*"
    }
  ]
}
```

Resource-based policy examples for AWS Elemental MediaTailor

To learn how to attach a resource-based policy to a channel, see [Create a channel using the MediaTailor console](#).

Topics

- [Anonymous access](#)
- [Cross-account access](#)

Anonymous access

Consider the following Allow policy. With this policy in effect, MediaTailor allows anonymous access to the `mediatailor:GetManifest` action on the channel resource in the policy. This occurs where *region* is the AWS Region, *accountID* is your AWS account ID, and *channelName* is the name of the channel resource.

Cross-account access

Consider the following Allow policy. With this policy in effect, MediaTailor allows the `mediatailor:GetManifest` action on the channel resource in the policy, across accounts. This occurs where *region* is the AWS Region, *accountID* is your AWS account ID, and *channelName* is the name of the channel resource.

AWS managed policies for AWS Elemental MediaTailor

To add permissions to users, groups, and roles, it is easier to use AWS managed policies than to write policies yourself. It takes time and expertise to [create IAM customer managed policies](#) that provide your team with only the permissions they need. To get started quickly, you can use our AWS managed policies. These policies cover common use cases and are available in your AWS account. For more information about AWS managed policies, see [AWS managed policies](#) in the *IAM User Guide*.

AWS services maintain and update AWS managed policies. You can't change the permissions in AWS managed policies. Services occasionally add additional permissions to an AWS managed policy to support new features. This type of update affects all identities (users, groups, and roles) where the policy is attached. Services are most likely to update an AWS managed policy when

a new feature is launched or when new operations become available. Services do not remove permissions from an AWS managed policy, so policy updates won't break your existing permissions.

Additionally, AWS supports managed policies for job functions that span multiple services. For example, the **ReadOnlyAccess** AWS managed policy provides read-only access to all AWS services and resources. When a service launches a new feature, AWS adds read-only permissions for new operations and resources. For a list and descriptions of job function policies, see [AWS managed policies for job functions](#) in the *IAM User Guide*.

AWS managed policy: AWSElementalMediaTailorFullAccess

You can attach the `AWSElementalMediaTailorFullAccess` policy to your IAM identities. It's useful for users who need to create and manage playback configurations and channel assembly resources, such as programs and channels. This policy grants permissions that allow full access to AWS Elemental MediaTailor. These users can create, update, and delete MediaTailor resources.

To view the permissions for this policy, see [AWSElementalMediaTailorFullAccess](#) in the *AWS Managed Policy Reference*.

AWS managed policy: AWSElementalMediaTailorReadOnly

You can attach the `AWSElementalMediaTailorReadOnly` policy to your IAM identities. It's useful for users who need to view playback configurations and channel assembly resources, such as programs and channels. This policy grants permissions that allow read-only access to AWS Elemental MediaTailor. These users can't create, update, or delete MediaTailor resources.

To view the permissions for this policy, see [AWSElementalMediaTailorReadOnly](#) in the *AWS Managed Policy Reference*.

MediaTailor updates to AWS managed policies

View details about updates to AWS managed policies for MediaTailor since this service began tracking these changes. For automatic alerts about changes to this page, subscribe to the RSS feed on the MediaTailor [Document history for AWS Elemental MediaTailor](#).

Change	Description	Date
MediaTailor added new managed policies	MediaTailor added the following managed policies:	November 24, 2021

Change	Description	Date
	• AWSElementalMediaTailorReadOnly • AWSElementalMediaTailorFullAccess	
MediaTailor started tracking changes	MediaTailor started tracking changes for its AWS managed policies.	November 24, 2021

Using service-linked roles for MediaTailor

AWS Elemental MediaTailor uses AWS Identity and Access Management (IAM) [service-linked roles](#). A service-linked role is a unique type of IAM role that is linked directly to MediaTailor. Service-linked roles are predefined by MediaTailor and include all the permissions that the service requires to call other AWS services on your behalf.

A service-linked role makes setting up MediaTailor easier because you don't have to manually add the necessary permissions. MediaTailor defines the permissions of its service-linked roles, and unless defined otherwise, only MediaTailor can assume its roles. The defined permissions include the trust policy and the permissions policy, and that permissions policy cannot be attached to any other IAM entity.

You can delete a service-linked role only after first deleting their related resources. This protects your MediaTailor resources because you can't inadvertently remove permission to access the resources.

For information about other services that support service-linked roles, see [AWS Services That Work with IAM](#) and look for the services that have **Yes** in the **Service-Linked Role** column. Choose a **Yes** with a link to view the service-linked role documentation for that service.

Supported Regions for MediaTailor service-linked roles

MediaTailor supports using service-linked roles in all of the regions where the service is available. For more information, see [AWS Regions and Endpoints](#).

Topics

- [Service-linked role permissions for MediaTailor](#)

- [Creating a service-linked role for MediaTailor](#)
- [Editing a service-linked role for MediaTailor](#)
- [Deleting a service-linked role for MediaTailor](#)

Service-linked role permissions for MediaTailor

MediaTailor uses the service-linked role named **AWSServiceRoleForMediaTailor** – MediaTailor uses this service-linked role to invoke CloudWatch to create and manage log groups, log streams, and log events. This service-linked role is attached to the following managed policy: **AWSMediaTailorServiceRolePolicy**.

The **AWSServiceRoleForMediaTailor** service-linked role trusts the following services to assume the role:

- `mediatailor.amazonaws.com`

The role permissions policy allows MediaTailor to complete the following actions on the specified resources:

- Action: `logs:PutLogEvents` on `arn:aws:logs:*:*:log-group:/aws/MediaTailor/*:log-stream:*`
- Action: `logs:CreateLogStream`, `logs:CreateLogGroup`, `logs:DescribeLogGroups`, `logs:DescribeLogStreams` on `arn:aws:logs:*:*:log-group:/aws/MediaTailor/*`

You must configure permissions to allow an IAM entity (such as a user, group, or role) to create, edit, or delete a service-linked role. For more information, see [Service-Linked Role Permissions](#) in the *IAM User Guide*.

Creating a service-linked role for MediaTailor

You don't need to manually create a service-linked role. When you enable session logging in the AWS Management Console, the AWS Command Line Interface (AWS CLI), or the AWS API, MediaTailor creates the service-linked role for you.

Important

This service-linked role can appear in your account if you completed an action in another service that uses the features supported by this role. Also, if you were using the MediaTailor

service before September 15, 2021, when it began supporting service-linked roles, then MediaTailor created the `AWSServiceRoleForMediaTailor` role in your account. To learn more, see [A New Role Appeared in My IAM Account](#).

If you delete this service-linked role, and then need to create it again, you can use the same process to recreate the role in your account. When you enable session logging, MediaTailor creates the service-linked role for you again.

You can also use the IAM console to create a service-linked role with the **MediaTailor** use case. In the AWS CLI or the AWS API, create a service-linked role with the `mediatailor.amazonaws.com` service name. For more information, see [Creating a Service-Linked Role](#) in the *IAM User Guide*. If you delete this service-linked role, you can use this same process to create the role again.

Editing a service-linked role for MediaTailor

MediaTailor does not allow you to edit the `AWSServiceRoleForMediaTailor` service-linked role. After you create a service-linked role, you cannot change the name of the role because various entities might reference the role. However, you can edit the description of the role using IAM. For more information, see [Editing a Service-Linked Role](#) in the *IAM User Guide*.

Deleting a service-linked role for MediaTailor

If you no longer need to use a feature or service that requires a service-linked role, we recommend that you delete that role. That way you don't have an unused entity that is not actively monitored or maintained. However, you must clean up the resources for your service-linked role before you can manually delete it.

Note

If the MediaTailor service is using the role when you try to clean up the resources, then the deletion might fail. If that happens, wait for a few minutes and try the operation again.

To clean up MediaTailor resources used by the `AWSServiceRoleForMediaTailor`

- Before you can delete the service-linked role created by MediaTailor for the log configuration, you must first *deactivate* all of the log configurations in your account. To deactivate a log configuration, set the **percent enabled** value to **0**. This turns off all session logging

the corresponding playback configuration. For more information, see [Deactivating a log configuration](#).

To manually delete the service-linked role using IAM

Use the IAM console, the AWS Command Line Interface (AWS CLI), or the AWS API to delete the `AWSServiceRoleForMediaTailor` service-linked role. For more information, see [Deleting a Service-Linked Role](#) in the *IAM User Guide*.

Troubleshooting AWS Elemental MediaTailor identity and access

Use the following information to help you diagnose and fix common issues that you might encounter when working with MediaTailor and IAM.

Topics

- [I am not authorized to perform an action in MediaTailor](#)
- [I am not authorized to perform `iam:PassRole`](#)
- [I want to allow people outside of my AWS account to access my MediaTailor resources](#)
- [MediaTailor cannot access my Amazon S3 bucket](#)

I am not authorized to perform an action in MediaTailor

If you receive an error that you're not authorized to perform an action, your policies must be updated to allow you to perform the action.

The following example error occurs when the `mateojackson` IAM user tries to use the console to view details about a fictional *my-example-widget* resource but doesn't have the fictional `mediatailor:GetWidget` permissions.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
mediatailor:GetWidget on resource: my-example-widget
```

In this case, the policy for the `mateojackson` user must be updated to allow access to the *my-example-widget* resource by using the `mediatailor:GetWidget` action.

If you need help, contact your AWS administrator. Your administrator is the person who provided you with your sign-in credentials.

I am not authorized to perform iam:PassRole

If you receive an error that you're not authorized to perform the `iam:PassRole` action, your policies must be updated to allow you to pass a role to MediaTailor.

Some AWS services allow you to pass an existing role to that service instead of creating a new service role or service-linked role. To do this, you must have permissions to pass the role to the service.

The following example error occurs when an IAM user named `marymajor` tries to use the console to perform an action in MediaTailor. However, the action requires the service to have permissions that are granted by a service role. Mary does not have permissions to pass the role to the service.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
iam:PassRole
```

In this case, Mary's policies must be updated to allow her to perform the `iam:PassRole` action.

If you need help, contact your AWS administrator. Your administrator is the person who provided you with your sign-in credentials.

I want to allow people outside of my AWS account to access my MediaTailor resources

You can create a role that users in other accounts or people outside of your organization can use to access your resources. You can specify who is trusted to assume the role. For services that support resource-based policies or access control lists (ACLs), you can use those policies to grant people access to your resources.

To learn more, consult the following:

- To learn whether MediaTailor supports these features, see [How AWS Elemental MediaTailor works with IAM](#).
- To learn how to provide access to your resources across AWS accounts that you own, see [Providing access to an IAM user in another AWS account that you own](#) in the *IAM User Guide*.
- To learn how to provide access to your resources to third-party AWS accounts, see [Providing access to AWS accounts owned by third parties](#) in the *IAM User Guide*.
- To learn how to provide access through identity federation, see [Providing access to externally authenticated users \(identity federation\)](#) in the *IAM User Guide*.

- To learn the difference between using roles and resource-based policies for cross-account access, see [Cross account resource access in IAM](#) in the *IAM User Guide*.

MediaTailor cannot access my Amazon S3 bucket

If MediaTailor returns errors when trying to access content from your Amazon S3 bucket, check the following common causes:

Amazon S3 permissions issues

Error symptoms:

- 502 BadGatewayException with Amazon S3 related messages
- 403 Forbidden errors in MediaTailor logs
- Content fails to load from Amazon S3 origin

Resolution:

1. Verify your MediaTailor service role has the required Amazon S3 permissions:
2. Check your Amazon S3 bucket policy allows MediaTailor access:

HTTPS access issues

Error symptoms:

- SSL/TLS handshake failures
- 502 Bad Gateway when accessing HTTPS Amazon S3 URLs

Resolution:

1. Ensure your Amazon S3 bucket policy requires HTTPS:
2. Verify MediaTailor configuration uses HTTPS URLs for Amazon S3 content.

Cross-account Amazon S3 access

Error symptoms:

- Access Denied errors when Amazon S3 bucket is in different account
- MediaTailor cannot assume cross-account roles

Resolution:

1. Update the Amazon S3 bucket policy in the content account to trust MediaTailor's service role
2. Add cross-account permissions to MediaTailor's service role
3. Ensure both accounts have proper trust relationships configured

Compliance validation for AWS Elemental MediaTailor

To learn whether an AWS service is within the scope of specific compliance programs, see [AWS services in Scope by Compliance Program](#) and choose the compliance program that you are interested in. For general information, see [AWS Compliance Programs](#).

You can download third-party audit reports using AWS Artifact. For more information, see [Downloading Reports in AWS Artifact](#).

Your compliance responsibility when using AWS services is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. For more information about your compliance responsibility when using AWS services, see [AWS Security Documentation](#).

Resilience in AWS Elemental MediaTailor

The AWS global infrastructure is built around AWS Regions and Availability Zones. AWS Regions provide multiple physically separated and isolated Availability Zones, which are connected with low-latency, high-throughput, and highly redundant networking. With Availability Zones, you can design and operate applications and databases that automatically fail over between zones without interruption. Availability Zones are more highly available, fault tolerant, and scalable than traditional single or multiple data center infrastructures.

For more information about AWS Regions and Availability Zones, see [AWS Global Infrastructure](#).

In addition to the AWS global infrastructure, MediaTailor offers several features to help support your data resiliency and backup needs.

Infrastructure security in MediaTailor

As a managed service, AWS Elemental MediaTailor is protected by AWS global network security. For information about AWS security services and how AWS protects infrastructure, see [AWS Cloud Security](#). To design your AWS environment using the best practices for infrastructure security, see [Infrastructure Protection](#) in *Security Pillar AWS Well-Architected Framework*.

You use AWS published API calls to access MediaTailor through the network. Clients must support the following:

- Transport Layer Security (TLS). We require TLS 1.2 and recommend TLS 1.3.
- Cipher suites with perfect forward secrecy (PFS) such as DHE (Ephemeral Diffie-Hellman) or ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). Most modern systems such as Java 7 and later support these modes.

You can call these API operations from any network location, but MediaTailor does support resource-based access policies, which can include restrictions based on the source IP address. You can also use MediaTailor policies to control access from specific Amazon Virtual Private Cloud (Amazon VPC) endpoints or specific VPCs. Effectively, this isolates network access to a given MediaTailor resource from only the specific VPC within the AWS network.

Cross-service confused deputy prevention

The confused deputy problem is a security issue where an entity that doesn't have permission to perform an action can coerce a more-privileged entity to perform the action. In AWS, cross-service impersonation can result in the confused deputy problem. Cross-service impersonation can occur when one service (the *calling service*) calls another service (the *called service*). The calling service can be manipulated to use its permissions to act on another customer's resources in a way it should not otherwise have permission to access. To prevent this, AWS provides tools that help you protect your data for all services with service principals that have been given access to resources in your account.

We recommend using the [aws:SourceArn](#) and [aws:SourceAccount](#) global condition context keys in resource policies to limit the permissions that AWS Elemental MediaTailor gives another service to the resource. If you use both global condition context keys, the `aws:SourceAccount` value and the account in the `aws:SourceArn` value must use the same account ID when used in the same policy statement.

The value of `aws:SourceArn` must be the playback configuration that MediaTailor publishes CloudWatch logs for in your Region and account. However, this only applies if you use the [MediaTailorLogger](#) role that lets MediaTailor publish Amazon CloudWatch logs to your account. This doesn't apply if you use a [service-linked role](#) to let MediaTailor publish the CloudWatch logs.

The most effective way to protect against the confused deputy problem is to use the `aws:SourceArn` global condition context key with the full ARN of the resource. If you don't know the full ARN of the resource or if you are specifying multiple resources, use the `aws:SourceArn` global context condition key with wildcards (*) for the unknown portions of the ARN. For example, `arn:aws:servicename::123456789012:*`.

The following example shows how you can use the `aws:SourceArn` and `aws:SourceAccount` global condition context keys in MediaTailor to prevent the confused deputy problem.

Logging and monitoring in MediaTailor

This section provides an overview of the options for logging and monitoring in AWS Elemental MediaTailor for security purposes. For more information about logging and monitoring in MediaTailor see [Monitoring and tagging AWS Elemental MediaTailor resources](#).

Monitoring is an important part of maintaining the reliability, availability, and performance of AWS Elemental MediaTailor and your AWS solutions. You should collect monitoring data from all of the parts of your AWS solution so that you can more easily debug a multi-point failure if one occurs. AWS provides several tools for monitoring your MediaTailor resources and responding to potential incidents:

Amazon CloudWatch Alarms

Using CloudWatch alarms, you watch a single metric over a time period that you specify. If the metric exceeds a given threshold, a notification is sent to an Amazon SNS topic or AWS Auto Scaling policy. CloudWatch alarms don't invoke actions because they are in a particular state. Rather, the state must have changed and been maintained for a specified number of periods. For more information, see [the section called "Monitoring with CloudWatch metrics"](#).

AWS CloudTrail logs

CloudTrail provides a record of actions taken by a user, role, or an AWS service in AWS Elemental MediaTailor. Using the information collected by CloudTrail, you can determine the request that

was made to MediaTailor, the IP address from which the request was made, who made the request, when it was made, and additional details. For more information, see [Recording AWS Elemental MediaTailor API calls](#).

AWS Trusted Advisor

Trusted Advisor draws upon best practices learned from serving hundreds of thousands of AWS customers. Trusted Advisor inspects your AWS environment and then makes recommendations when opportunities exist to save money, improve system availability and performance, or help close security gaps. All AWS customers have access to five Trusted Advisor checks. Customers with a Business or Enterprise support plan can view all Trusted Advisor checks.

For more information, see [AWS Trusted Advisor](#).

Monitoring and tagging AWS Elemental MediaTailor resources

Monitoring is an important part of maintaining the reliability, availability, and performance of AWS Elemental MediaTailor and your other AWS solutions. AWS provides the following monitoring tools to watch MediaTailor, report when something is wrong, and take automatic actions when appropriate:

- *Amazon CloudWatch* monitors your AWS resources and the applications that you run on AWS in real time. You can collect and track metrics, create customized dashboards, and set alarms that notify you or take actions when a specified metric reaches a threshold that you specify. For example, you can have CloudWatch track CPU usage or other metrics of your Amazon EC2 instances and automatically launch new instances when needed. For more information, see the [Amazon CloudWatch User Guide](#).
- *Amazon CloudWatch Logs* enables you to monitor, store, and access your log files from all interactions with your ad decision server (ADS). AWS Elemental MediaTailor emits logs for ad requests, redirects, responses, and reporting requests and responses. Errors from the ADS and origin servers are also emitted to log groups in Amazon CloudWatch. MediaTailor also provides information about skipped ads and the reasons they were skipped. For more information, see [Ad skipping troubleshooting](#). You can also archive your log data in highly durable storage. For general information, see the [Amazon CloudWatch Logs User Guide](#). For information on the ADS logs and how to access them for analysis through Amazon CloudWatch Logs Insights, see [AWS Elemental MediaTailor ADS log analysis in Amazon CloudWatch Logs Insights](#).

Topics

- [Viewing AWS Elemental MediaTailor logs](#)
- [Monitoring AWS Elemental MediaTailor with Amazon CloudWatch metrics](#)
- [Recording AWS Elemental MediaTailor API calls](#)
- [Receiving AWS Elemental MediaTailor channel assembly alerts](#)
- [Tagging AWS Elemental MediaTailor resources](#)
- [Monitoring AWS media services with workflow monitor](#)

Viewing AWS Elemental MediaTailor logs

MediaTailor emits logs that describe a variety of milestones and activities in channels and playback configurations. You can use these logs to gain visibility into your workflow and to troubleshoot issues with the service. The following topics describe logs and logging options.

Topics

- [AWS Elemental MediaTailor ADS logs description and event types](#)
- [AWS Elemental MediaTailor manifest logs description and event types](#)
- [AWS Elemental MediaTailor transcode logs description and event types](#)
- [Using vended logs to send AWS Elemental MediaTailor logs](#)
- [Writing AWS Elemental MediaTailor logs directly to Amazon CloudWatch Logs](#)
- [Controlling the volume of AWS Elemental MediaTailor logs](#)
- [Filtering AWS Elemental MediaTailor logs and events](#)
- [Generating AWS Elemental MediaTailor debug logs](#)

AWS Elemental MediaTailor ADS logs description and event types

The following sections describe the logs that MediaTailor emits to describe events with the ad decision server (ADS). These are AdDecisionServerInteractions logs.

Topics

- [AdDecisionServerInteractions events](#)
- [ADS log description](#)
- [ADS log JSON schema](#)

AdDecisionServerInteractions events

The following events are emitted during MediaTailor interactions with the ad decision server (ADS).

Log	Description
AD_MARKER_FOUND	MediaTailor found an ad marker in the manifest.

Log	Description
BEACON_FIRED	A tracking beacon was fired to report ad events. In server-side reporting mode (default) , MediaTailor fires the beacon. In client-side reporting mode, the playback device fires the beacon.
EMPTY_VAST_RESPONSE	The ADS returned an empty VAST response containing zero ads.
EMPTY_VMAP_RESPONSE	The ADS returned an empty VMAP response.
ERROR_ADS_INVALID_RESPONSE	The ADS returned a non-200 status code.
ERROR_ADS_IO	MediaTailor encountered an error while trying to communicate with the ADS.
ERROR_ADS_RESPONSE_PARSE	MediaTailor encountered an error while parsing the ADS response.
ERROR_ADS_RESPONSE_UNKNOWN_ROOT_ELEMENT	The ADS response contains an invalid root element.
ERROR_ADS_TIMEOUT	The MediaTailor request to the ADS timed out.
ERROR_DISALLOWED_HOST	The ADS host is not allowed.
ERROR_FIRING_BEACON_FAILED	MediaTailor failed at firing the tracking beacon.
ERROR_PERSONALIZATION_DISABLED	Ad insertion is disabled for this session.
ERROR_UNKNOWN	MediaTailor encountered an unknown error during the ADS request.
ERROR_UNKNOWN_HOST	The ADS host is unknown.
ERROR_VAST_INVALID_MEDIA_FILE	The VAST Ad has an invalid or missing MediaFile element.

Log	Description
ERROR_VAST_INVALID_VAST_AD_TAG_URI	The VAST response contains an invalid VASTAdTagURI .
ERROR_VAST_MISSING_CREATIVES	The VAST Ad contains zero or multiple Creatives elements. Exactly one Creatives element is required.
ERROR_VAST_MISSING_IMPRESSION	The VAST Ad contains zero Impression elements. At least one Impression element is required.
ERROR_VAST_MISSING_MEDIAFILES	The VAST Ad contains zero or multiple MediaFiles elements. Exactly one MediaFiles element is required.
ERROR_VAST_MISSING_OVERLAYS	MediaTailor didn't receive any non-linear creatives from the ad server.
ERROR_VAST_MULTIPLE_LINEAR	The VAST Ad contains multiple Linear elements.
ERROR_VAST_MULTIPLE_TRACKING_EVENTS	The VAST Ad contains multiple TrackingEvents elements.
ERROR_VAST_REDIRECT_EMPTY_RESPONSE	The VAST redirect request returned an empty response.
ERROR_VAST_REDIRECT_FAILED	The VAST redirect request encountered an error.
ERROR_VAST_REDIRECT_MULTIPLE_VAST	The VAST redirect request returned multiple ads.
FILLED_AVAIL	MediaTailor successfully filled the avail.
FILLED_OVERLAY_AVAIL	MediaTailor successfully filled the overlay avail.

Log	Description
INTERSTITIAL_VOD_FAILURE	The ADS request or response encountered a problem while filling interstitial avails for the VOD playlist. No ads will be inserted.
INTERSTITIAL_VOD_SUCCESS	MediaTailor successfully filled interstitial avails for the VOD playlist.
MAKING_ADS_REQUEST	MediaTailor is requesting advertisements from the ADS.
MODIFIED_TARGET_URL	MediaTailor modified the outbound target URL.
NON_AD_MARKER_FOUND	MediaTailor found a non-actionable ad marker in the manifest.
PRE_ADS_REQUEST_FUNCTION_COMPLETED	An individual function in the pre-ads request hook completed. This is an opt-in event type.
PRE_ADS_REQUEST_FUNCTION_ERROR	An individual function in the pre-ads request hook failed.
PRE_ADS_REQUEST_HOOK_ERROR	The pre-ads request hook execution failed.
PRE_ADS_REQUEST_HOOK_SUMMARY	Summary of the pre-ads request hook execution, including success or error status. This is an opt-in event type.
RAW_ADS_RESPONSE	MediaTailor received a raw ADS response.
REDIRECTED_VAST_RESPONSE	MediaTailor received a VAST response after following the VAST redirect.
VAST_REDIRECT	The VAST ad response contains a redirect.
VAST_RESPONSE	MediaTailor received a VAST response.

Log	Description
VOD_TIME_BASED_AVAIL_PLAN_SUCCESS	MediaTailor successfully created a time-based avail plan for the VOD template.
VOD_TIME_BASED_AVAIL_PLAN_VAST_RESPONSE_FOR_OFFSET	MediaTailor is creating a time-based avail plan for the VOD template. MediaTailor received a VAST response for the time offset.
VOD_TIME_BASED_AVAIL_PLAN_WARNING_NO_ADVERTISEMENTS	The ADS request or response encountered a problem while creating a time-based avail plan for the VOD template. No ads will be inserted.
WARNING_NO_ADVERTISEMENTS	MediaTailor encountered a problem while filling the avail. No ads are inserted.
WARNING_URL_VARIABLE_SUBSTITUTION_FAILED	MediaTailor can't substitute dynamic variables in the ADS URL. Check the URL configuration.
WARNING_VPAID_AD_DROPPED	A VPAID ad dropped due to a missing slate, or the session uses server-side reporting.

ADS log description

This section describes the structure and contents of the ADS log description. To explore on your own in a JSON editor, use the listing at [the section called “ADS log JSON schema”](#).

Each event in the ADS log contains the standard fields that are generated by CloudWatch Logs. For information, see [Analyze log data with CloudWatch Logs insights](#).

ADS logs properties

This section describes the properties of the ADS logs.

Property	Type	Required	Description
adsRequestUrl	string	false	The full URL of the ADS request made by MediaTailor.
avail	object of type avail	false	Information about an avail that MediaTailor fills with ads. Currently, for the FILLED_AVAIL event type, this is the plan created by MediaTailor when it first encounters the avail. How the avail is eventually filled may vary from this plan, depending on how the content plays out.
awsAccountId	string	true	The AWS account ID for the MediaTailor configuration that was used for the session.
customerId	string	true	The hashed version of the AWS account ID, which you can use to correlate multiple log entries.
eventDescription	string	true	A short description of the event that triggered this log message,

Property	Type	Required	Description
			provided by the MediaTailor service. By default, this is empty. Example: Got VAST response.
eventTimestamp	string	true	The date and time of the event.
eventType	string	true	The code for the event that triggered this log message. Example: VAST_RESPONSE .
errorType	string	false	Structured error classification. Present on PRE_ADS_REQUEST_HO_OK_ERROR and PRE_ADS_REQUEST_FUNCTION_ERROR events. Values: INTERNAL_ERROR , SYNTAX_ERROR , RESOURCE_LIMIT_ERROR , RESTRICTION_ERROR , TIMEOUT_ERROR , VALIDATION_ERROR .

Property	Type	Required	Description
cause	string	false	Error detail string describing the specific failure. Present on PRE_ADS_REQUEST_H0_OK_ERROR and PRE_ADS_REQUEST_FUNCTION_ERROR events.
eventId	string	false	Unique identifier per hook invocation. Use this field to correlate hook-level and function-level events for the same execution. Present on Functions events only.
executionTimeMs	integer	false	Execution time in milliseconds. Present on Functions events only.

Property	Type	Required	Description
functionId	string	false	The identifier of the function. On hook events, this is the function configured on the hook. On function events, this is the individual function within the executor. Present on Functions events only.
functionType	string	false	The type of function: CUSTOM_OUTPUT , HTTP_REQUEST , SEQUENTIAL_EXECUTOR , or CONCURRENT_EXECUTOR . Present on function-level events only.

Property	Type	Required	Description
http	object	false	HTTP request and response details. Present on function-level events for HTTP_REQUEST functions only. Contains request (url, method, headers, body), responseTimeMs , response (statusCode, headers, body), and truncated (true when request or response body was truncated to fit within log event size limits).
input	object	false	Before-values of variables targeted by output expressions. Present on function-level events only.
output	object	false	Values produced by the function. Present on PRE_ADS_REQUEST_FUNCTION_COMPLETED events only.

Property	Type	Required	Description
status	string	false	Hook execution outcome: SUCCESS or ERROR. Present on PRE_ADS_REQUEST_HO OK_SUMMARY events only.
originId	string	true	The configuration name from the MediaTailor configuration. This is different from the video content source, which is also part of the configuration.
prefetchScheduleName	string	false	The name of the prefetch schedule associated with this ad event.
requestHeaders	array of type requestheaders	false	The headers that MediaTailor included with the ADS request. Typically, the logs include these when a request to the ADS fails, to help with troubleshooting.

Property	Type	Required	Description
requestId	string	true	The MediaTailor request ID, which you can use to correlate multiple log entries for the same request.
sessionId	string	true	The unique numeric identifier that MediaTailor assigned to the player session. All requests that a player makes for a session have the same session ID. Example: e039fd39-09f0-46b2-aca9-9871cc116cde .
sessionType	string (legal values: [DASH, HLS])	true	The player's stream type.
vastAd	object of type vastAd	false	Information about a single ad parsed from the VAST response.
vastResponse	object of type vastResponse	false	Information about the VAST response that MediaTailor received from the ADS.

Property	Type	Required	Description
<code>vodCreativeOffsets</code>	object of type vodCreativeOffsets	false	A map that indicates the time offsets in the manifest where MediaTailor will insert avails, based on the VMAP response.
<code>vodVastResponseTimeOffset</code>	number	false	The VMAP specific time offset for VOD ad insertion.

adContent

This section describes the properties of the ADS logs adContent.

ADS Logs adContent Properties

Property	Type	Required	Description
<code>adPlaylistUri</code>	object of type adPlaylistUri	false	The mapping from the origin manifest for a variant to the ad manifest for the variant. For DASH, this contains a single entry, because all variants are represented in a single DASH manifest.

adPlaylistUri

This section describes the properties of the ADS logs adPlaylistUri.

ADS Logs adPlaylistUris Properties

Property	Type	Required	Description
<any string>	string	false	The URL of the ad manifest for the specific variant.

avail

This section describes the properties of the ADS logs avail.

ADS Logs avail Properties

Property	Type	Required	Description
availId	string	true	The unique identifier for this avail. For HLS, this is the media sequence number where the avail begins. For DASH, this is the period ID.
creativeAds	array of type creativeAd	true	The ads that MediaTailor inserted into the avail.
fillRate	number	true	The rate at which the ads fill the avail duration, from 0.0 (for 0%) to 1.0 (for 100%).
filledDuration	number	true	The sum of the durations of all the ads inserted into the avail.

Property	Type	Required	Description
numAds	number	true	The number of ads that MediaTailor inserted into the avail.
originAvailDuration	number	true	The duration of the avail as specified in the content stream from the origin (CUE_OUT or SCTE).
skippedAds	array of type skippedAd	false	The ads that MediaTailor didn't insert, for reasons like TRANSCODE_IN_PROGRESS and TRANSCODE_ERROR . For a list of skipped ad reasons, see Ad skipping troubleshooting.
slateAd	object of type slateAd	true	Information about the slate ad, which MediaTailor uses to fill any unfilled segments in the avail.

creativeAd

This section describes the properties of the ADS logs creativeAd.

ADS Logs creativeAd Properties

Property	Type	Required	Description
adContent	object of type adContent	true	Information about the content of the inserted ad.
creativeUniqueId	string	true	The unique identifier for the ad, used as a key for transcoding. This is the ID field for the creative in the VAST response, if available. Otherwise, it's the mezzanine URL of the ad.
trackingEvents	object of type trackingEvents	true	The tracking beacon URLs for the various tracking events for the ad. The keys are the event names, and the values are a list of beacon URLs.
transcodedAdDuration	number	true	The duration of the ad, calculated from the transcoded asset.
uri	string	true	The URL of the mezzanine version of the ad, which is the input to the transcoder.

Property	Type	Required	Description
vastDuration	number	true	The duration of the ad, as parsed from the VAST response.

requestheaders

This section describes the properties of the ADS logs requestheaders.

ADS Logs requestheaders Properties

Property	Type	Required	Description
name	string	true	The name of the header.
value	string	true	The value of the header.

skippedAd

This section describes the properties of the ADS logs skippedAd.

ADS Logs skippedAd Properties

Property	Type	Required	Description
adMezzanineUrl	string	true	The mezzanine URL of the skipped ad.
creativeUniqueId	string	true	The unique identifier for the ad, used as a key for transcoding. This is the ID field for the creative in the VAST response, if available. Otherwise

Property	Type	Required	Description
			, it's the mezzanine URL of the ad.
skippedReason	string	true	The code that indicates why the ad wasn't inserted. Example: TRANSCODE_IN_PROGRESS . For a list of skipped ad reasons, see Ad skipping troubleshooting.
transcodeAdDuration	number	false	The duration of the ad, calculated from the transcoded asset.
vastDuration	number	true	The duration of the ad, as parsed from the VAST response.

slateAd

This section describes the properties of the ADS logs slateAd.

ADS Logs slateAd Properties

Property	Type	Required	Description
adContent	object of type adContent	true	Information about the content of the inserted ad.
creativeUniqueId	string	true	The unique identifier for the ad, used as a key for transcodi

Property	Type	Required	Description
			ng. This is the ID field for the creative in the VAST response, if available. Otherwise , it's the mezzanine URL of the ad.
transcode dAdDuration	number	true	The duration of the ad, calculated from the transcoded asset.
uri	string	true	The URL of the mezzanine version of the ad, which is the input to the transcoder.

trackingEvents

This section describes the properties of the ADS logs trackingEvents.

ADS Logs trackingEvents Properties

Property	Type	Required	Description
<any string>	array of type string	false	The list of beacon URLs for the specified tracking event (impression, complete, and so on)

vastAd

This section describes the properties of the ADS logs vastAd.

ADS Logs vastAd Properties

Property	Type	Required	Description
adSystem	string	true	The value of the AdSystem tag in the VAST response.
adTitle	string	true	The media files that are available for the ad in the VAST response.
creativeAdId	string	true	The value of the adId attribute of the Creative tag in the VAST response.
creativeId	string	true	The value of the id attribute of the Creative tag in the VAST response.
duration	number	true	The approximate duration of the ad, based on the duration tag in the linear element of the VAST response.
trackingEvents	object of type trackingEvents	true	The tracking beacon URLs for the various tracking events for the ad. The keys are the event names, and the values are a list of beacon URLs.

Property	Type	Required	Description
vastAdId	string	true	The value of the id attribute of the Ad tag in the VAST response
vastAdTagUri	string	false	The VMAP-specific redirect URI for an ad.
vastMediaFiles	array of type vastMediaFile	true	The list of available media files for the ad in the VAST response.

vastMediaFile

This section describes the properties of the ADS logs vastMediaFile.

ADS Logs vastMediaFile Properties

Property	Type	Required	Description
apiFramework	string	true	The API framework needed to manage the media file. Example: VPAID.
bitrate	number	true	The bitrate of the media file.
delivery	string	true	The protocol used for the media file, set to either progressive or streaming.
height	number	true	The pixel height of the media file.

Property	Type	Required	Description
id	string	true	The value of the id attribute of the <code>MediaFile</code> tag.
type	string	true	The MIME type of the media file, taken from the type attribute of the <code>MediaFile</code> tag.
uri	string	true	The URL of the mezzanine version of the ad, which is the input to the transcoder.
width	number	true	The pixel width of the media file.

vastResponse

This section describes the properties of the ADS logs `vastResponse`.

ADS Logs `vastResponse` Properties

Property	Type	Required	Description
errors	array of type string	true	The error URLs parsed from the <code>Error</code> tags in the VAST response.
vastAds	array of type vastAd	true	The ads parsed from the VAST response.
version	string	true	The VAST specification version, parsed

Property	Type	Required	Description
			from the version attribute of the VAST tag in the response.

vodCreativeOffsets

This section describes the properties of the ADS logs vodCreativeOffsets.

ADS Logs vodCreativeOffsets Properties

Property	Type	Required	Description
<any string>	array of type vodCreativeOffset	false	A mapping from a time offset in the manifest to a list of ads to insert at this time.

vodCreativeOffset

This section describes the properties of the ADS logs vodCreativeOffset.

ADS Logs vodCreativeOffset Properties

Property	Type	Required	Description
adContent	object of type adContent	true	Information about the content of the inserted ad.
creativeUniqueId	string	true	The unique identifier for the ad, used as a key for transcoding. This is the ID field for the creative in the VAST response, if available. Otherwise

Property	Type	Required	Description
			, it's the mezzanine URL of the ad.
trackingEvents	object of type trackingEvents	true	The tracking beacon URLs for the various tracking events for the ad. The keys are the event names, and the values are a list of beacon URLs.
transcodeAdDuration	number	true	The duration of the ad, calculated from the transcoded asset.
uri	string	true	The URL of the mezzanine version of the ad, which is the input to the transcoder.
vastDuration	number	true	The duration of the ad, as parsed from the VAST response.

ADS log JSON schema

The following lists the JSON schema for the AWS Elemental MediaTailor ADS log.

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "$id": "http://amazon.com/elemental/midas/mms/adsLogSchema.json",
  "type": "object",
  "title": "AWS Elemental MediaTailor ADS Log JSON Schema",
  "required": [
    "eventType",
    "eventTimestamp",
```

```

    "requestId",
    "sessionType",
    "eventDescription",
    "awsAccountId",
    "customerId",
    "originId",
    "sessionId"
  ],
  "additionalProperties": false,
  "properties": {
    "eventType": {
      "$id": "#/properties/eventType",
      "type": "string",
      "description": "The code for the event that triggered this log message. Example:
<code>VAST_RESPONSE</code>.",
      "examples": [
        "FILLED_AVAIL"
      ]
    },
    "eventTimestamp": {
      "$id": "#/properties/eventTimestamp",
      "type": "string",
      "description": "The date and time of the event.",
      "examples": [
        "1970-01-01T00:00:00Z"
      ],
      "format": "date-time"
    },
    "requestId": {
      "$id": "#/properties/requestId",
      "type": "string",
      "description": "The MediaTailor request ID, which you can use to correlate
multiple log entries for the same request.",
      "examples": [
        "c7c7ae8c-a61e-44e0-8efd-7723995337a1"
      ],
      "pattern": "^(.*)$"
    },
    "sessionType": {
      "$id": "#/properties/sessionType",
      "type": "string",
      "enum": [
        "HLS",
        "DASH"
      ]
    }
  }
}

```



```

    ],
    "description": "The player's stream type."
  },
  "eventDescription": {
    "$id": "#/properties/eventDescription",
    "type": "string",
    "description": "A short description of the event that triggered this log message,
provided by the MediaTailor service. By default, this is empty. Example: <code>Got
VAST response</code>.",
    "default": "",
    "examples": [
      "Got VAST response"
    ],
    "pattern": "^(.*)$"
  },
  "awsAccountId": {
    "$id": "#/properties/awsAccountId",
    "type": "string",
    "description": "The AWS account ID for the MediaTailor configuration that was
used for the session."
  },
  "customerId": {
    "$id": "#/properties/customerId",
    "type": "string",
    "description": "The hashed version of the AWS account ID, which you can use to
correlate multiple log entries.",
    "pattern": "^(.*)$"
  },
  "originId": {
    "$id": "#/properties/originId",
    "type": "string",
    "description": "The configuration name from the MediaTailor configuration. This
is different from the video content source, which is also part of the configuration.",
    "examples": [
      "external-canary-dash-serverside-reporting-onebox"
    ],
    "pattern": "^(.*)$"
  },
  "sessionId": {
    "$id": "#/properties/sessionId",
    "type": "string",
    "description": "The unique numeric identifier that MediaTailor assigned to the
player session. All requests that a player makes for a session have the same session
ID. Example: <code>e039fd39-09f0-46b2-aca9-9871cc116cde</code>.",

```

```

    "examples": [
      "120b9873-c007-40c8-b3db-0f1bd194970b"
    ],
    "pattern": "^(.*)$"
  },
  "avail": {
    "$id": "#/properties/avail",
    "type": "object",
    "title": "avail",
    "description": "Information about an avail that MediaTailor fills with ads.
Currently, for the <code>FILLED_AVAIL</code> event type, this is the plan created by
MediaTailor when it first encounters the avail. How the avail is eventually filled may
vary from this plan, depending on how the content plays out. ",
    "required": [
      "creativeAds",
      "originAvailDuration",
      "filledDuration",
      "fillRate",
      "driftMillisecondsAtAvailStart",
      "numAds",
      "slateAd",
      "availId"
    ],
    "additionalProperties": false,
    "properties": {
      "originAvailDuration": {
        "$id": "#/properties/avail/originAvailDuration",
        "type": "number",
        "description": "The duration of the avail as specified in the content stream
from the origin (<code>CUE_OUT</code> or <code>SCTE</code>).",
      },
      "filledDuration": {
        "$id": "#/properties/avail/filledDuration",
        "type": "number",
        "description": "The sum of the durations of all the ads inserted into the
avail."
      },
      "fillRate": {
        "$id": "#/properties/avail/fillRate",
        "type": "number",
        "description": "The rate at which the ads fill the avail duration, from 0.0
(for 0%) to 1.0 (for 100%).",
      },
      "driftMillisecondsAtAvailStart": {

```

```

    "$id": "#/properties/avail/driftMillisecondsAtAvailStart",
    "type": "number",
    "description": "The cumulative drift at the beginning of this avail. A
positive value implies that we are moving away from the live edge, a negative value
implies that we are moving towards the live edge."
  },
  "creativeAds": {
    "$id": "#/properties/avail/creativeAds",
    "type": "array",
    "description": "The ads that MediaTailor inserted into the avail.",
    "items": {
      "type": "object",
      "title": "creativeAd",
      "description": "Information about a single inserted ad.",
      "required": [
        "uri",
        "creativeUniqueId",
        "adSystem",
        "adContent",
        "trackingEvents",
        "vastDuration",
        "transcodedAdDuration"
      ],
      "additionalProperties": false,
      "properties": {
        "uri": { "$ref": "#/definitions/adMezzanineUri" },
        "creativeUniqueId": { "$ref": "#/definitions/creativeUniqueId" },
        "adSystem": { "$ref": "#/definitions/adSystem" },
        "adContent": { "$ref": "#/definitions/adContent" },
        "trackingEvents": { "$ref": "#/definitions/trackingEvents" },
        "vastDuration": { "$ref": "#/definitions/vastDuration" },
        "transcodedAdDuration": { "$ref": "#/definitions/transcodedAdDuration" }
      }
    }
  },
  "numAds": {
    "$id": "#/properties/avail/numAds",
    "type": "number",
    "description": "The number of ads that MediaTailor inserted into the avail."
  },
  "slateAd": {
    "$id": "#/properties/avail/slateAd",
    "type": ["object", "null"],
    "title": "slateAd",

```

```

    "description": "Information about the slate ad, which MediaTailor uses to
fill any unfilled segments in the avail.",
    "additionalProperties": false,
    "required": [
        "uri",
        "creativeUniqueId",
        "adContent",
        "transcodedAdDuration"
    ],
    "properties": {
        "uri": { "$ref": "#/definitions/adMezzanineUri" },
        "creativeUniqueId": { "$ref": "#/definitions/creativeUniqueId" },
        "adContent": { "$ref": "#/definitions/adContent" },
        "transcodedAdDuration": { "$ref": "#/definitions/transcodedAdDuration" }
    }
},
"availId": {
    "$id": "#/properties/avail/availId",
    "type": "string",
    "description": "The unique identifier for this avail. For HLS, this is the
media sequence number where the avail begins. For DASH, this is the period ID."
},
"skippedAds": {
    "$id": "#/properties/avail/skippedAds",
    "type": "array",
    "description": "The ads that MediaTailor didn't insert, for reasons like
<code>TRANSCODE_IN_PROGRESS</code> and <code>TRANSCODE_ERROR</code>.",
    "items": {
        "type": "object",
        "title": "skippedAd",
        "description": "Information about a single skipped ad.",
        "required": [
            "creativeUniqueId",
            "adMezzanineUrl",
            "skippedReason",
            "vastDuration"
        ],
        "properties": {
            "creativeUniqueId": { "$ref": "#/definitions/creativeUniqueId" },
            "adMezzanineUrl": {
                "type": "string",
                "description": "The mezzanine URL of the skipped ad."
            }
        }
    },
    "additionalProperties": false,
    "properties": {
        "creativeUniqueId": { "$ref": "#/definitions/creativeUniqueId" },
        "adMezzanineUrl": {
            "type": "string",
            "description": "The mezzanine URL of the skipped ad."
        }
    }
},

```

```

    "skippedReason": {
      "type": "string",
      "description": "The code that indicates why the ad wasn't inserted.
Example: <code>TRANSCODE_IN_PROGRESS</code>."
    },
    "vastDuration": { "$ref": "#/definitions/vastDuration" },
    "transcodedAdDuration": { "$ref": "#/definitions/transcodedAdDuration" },
    "targetVariant": {
      "type": "object",
      "title": "targetVariant",
      "description": "The target variant of the source content. This key is
present when an ad wasn't inserted due to the source content containing a variant that
could not match to any variants present in this ad.",
      "required": [
        "mediaProtocol",
        "mediaType",
        "bitrate",
        "mediaResolution",
        "codecs"
      ],
      "additionalProperties": false,
      "properties": {
        "mediaProtocol": {
          "type": "string",
          "description": "The media protocol of this variant, such as HLS.",
          "enum": [
            "HLS",
            "DASH"
          ]
        },
        "mediaType": {
          "type": "array",
          "description": "The media type of this variant, such as VIDEO.",
          "items": {
            "type": "string",
            "enum": [
              "VIDEO",
              "AUDIO",
              "SUBTITLES",
              "TRICK_PLAY"
            ]
          },
          "description": "Media type, such as VIDEO."
        }
      }
    },
  },

```

```

        "bitrate": {
            "$ref": "#/definitions/bitrate"
        },
        "mediaResolution": {
            "type": "object",
            "title": "mediaResolution",
            "description": "The media resolution of this variant.",
            "required": [
                "width",
                "height"
            ],
            "additionalProperties": false,
            "properties": {
                "width": {
                    "$ref": "#/definitions/width"
                },
                "height": {
                    "$ref": "#/definitions/height"
                }
            }
        },
        "codecs": {
            "type": "array",
            "description": "The codecs of this variant.",
            "items": {
                "type": "string",
                "description": "Codec, such as avc1."
            }
        }
    },
    "adBreakTrackingEvents": {
        "$id": "#properties/avail/adBreakTrackingEvents",
        "type": "object",
        "title": "adBreakTrackingEvents",
        "description": "VMAP/ad break tracking events and corresponding URL",
        "properties": {
            "items": {
                "$ref": "#/definitions/adBreakTrackingEvents"
            }
        }
    }
}

```

```

    }
  }
},

"vastResponse": {
  "$id": "#/properties/vastResponse",
  "type": "object",
  "title": "vastResponse",
  "description": "Information about the VAST response that MediaTailor received
from the ADS.",
  "required": [
    "version",
    "vastAds",
    "errors",
    "nonLinearAdsList"
  ],
  "additionalProperties": false,
  "properties": {
    "version": {
      "$id": "#/properties/vastResponse/version",
      "type": "string",
      "description": "The VAST specification version, parsed from the
<code>version</code> attribute of the <code>VAST</code> tag in the response.",
      "examples": [
        "3.0"
      ],
      "pattern": "^(.*)$"
    },
    "vastAds": {
      "$id": "#/properties/vastResponse/vastAds",
      "type": "array",
      "description": "The ads parsed from the VAST response.",
      "items": {
        "$ref": "#/definitions/vastAd"
      }
    },
    "errors": {
      "$id": "#/properties/vastResponse/errors",
      "type": "array",
      "description": "The error URLs parsed from the <code>Error</code> tags in the
VAST response.",
      "items": {
        "type": "string",
        "description": "A single error URL."
      }
    }
  }
}

```

```

    }
  },
  "nonLinearAdsList": {
    "$id": "#/properties/vastResponse/nonLinearAds",
    "type": "array",
    "description": "A list of NonLinearAds as they are read from the VAST
response.",
    "items": {
      "$ref": "#/definitions/nonLinearAds"
    }
  }
},

"vastAd": {
  "$ref": "#/definitions/vastAd"
},

"vodVastResponseTimeOffset": {
  "$id": "#/properties/vodVastResponseTimeOffset",
  "type": "number",
  "description": "The VMAP specific time offset for VOD ad insertion.",
  "examples": [
    5.0
  ]
},

"vodCreativeOffsets": {
  "$id": "#/properties/vodCreativeOffsets",
  "type": "object",
  "title": "vodCreativeOffsets",
  "description": "A map that indicates the time offsets in the manifest where
MediaTailor will insert avails, based on the VMAP response.",
  "additionalProperties": {
    "type": "array",
    "$id": "#/properties/vodCreativeOffsets/entry",
    "description": "A mapping from a time offset in the manifest to a list of ads
to insert at this time.",
    "items": {
      "type": "object",
      "$id": "#/properties/vodCreativeOffsets/entry/items",
      "title": "vodCreativeOffset",
      "description": "The list of ads to insert at the specified time offset.",
      "additionalProperties": false,

```



```

    "required": [
      "uri",
      "creativeUniqueId",
      "vastDuration",
      "transcodedAdDuration",
      "adContent",
      "trackingEvents"
    ],
    "properties": {
      "uri": { "$ref": "#/definitions/adMezzanineUri" },
      "creativeUniqueId": { "$ref": "#/definitions/creativeUniqueId" },
      "vastDuration": { "$ref": "#/definitions/vastDuration" },
      "transcodedAdDuration": { "$ref": "#/definitions/transcodedAdDuration" },
      "adContent": { "$ref": "#/definitions/adContent" },
      "trackingEvents": { "$ref": "#/definitions/trackingEvents" }
    }
  }
},

"adsRequestUrl": {
  "$id": "#/properties/adsRequestUrl",
  "type": "string",
  "description": "The full URL of the ADS request made by MediaTailor."
},
"adMarkers": {
  "$id": "#/properties/adMarkers",
  "type": "string",
  "description": "Found Ad Marker in the Manifest."
},
"segmentationUpid": {
  "$id": "#/properties/segmentationUpid",
  "type": "string",
  "description": "Value of segmentation upid parsed from ad markers in manifest."
},
"segmentationTypeId": {
  "$id": "#/properties/segmentationTypeId",
  "type": "integer",
  "description": "Value of segmentation typeId parsed from ad markers in manifest."
},
"requestHeaders": {
  "$id": "#/properties/requestHeaders",
  "type": "array",

```

```

    "description": "The headers that MediaTailor included with the ADS request. Typically, the logs include these when a request to the ADS fails, to help with troubleshooting.",
    "items": {
      "type": "object",
      "title": "requestheaders",
      "description": "The name and value for a single header included in the ADS request.",
      "required": [
        "name",
        "value"
      ],
      "additionalProperties": false,
      "properties": {
        "name": {
          "type": "string",
          "description": "The name of the header."
        },
        "value": {
          "type": "string",
          "description": "The value of the header."
        }
      }
    }
  },
  "originalTargetUrl": {
    "$id": "#/properties/originalTargetUrl",
    "type": "string",
    "description": "The old URL to which MediaTailor was going to make a request."
  },
  "prefetchScheduleName": {
    "$id": "#/properties/prefetchScheduleName",
    "type": "string",
    "description": "The name of the prefetch schedule associated with this ad event.",
    "examples": [
      "PrefetchScheduleName"
    ],
    "pattern": "^(.*)$"
  },
  "updatedTargetUrl": {
    "$id": "#/properties/updatedTargetUrl",
    "type": "string",

```

```

    "description": "The new URL to which MediaTailor is making a request."
  },

  "rawAdsResponse": {
    "$id": "#/properties/rawAdsResponse",
    "type": "string",
    "description": "Paginated ADS response as it's exactly returned to MediaTailor."
  },
  "rawAdsResponseIndex": {
    "$id": "#/properties/rawAdsResponseIndex",
    "type": "integer",
    "description": "Integer value denoting this rawAdsResponse's index into the
full ADS response. This value is used to order the paginated messages for this ADS
response."
  },

  "eventId": {
    "$id": "#/properties/eventId",
    "type": "string",
    "description": "Unique identifier per hook invocation. Use this field to
correlate hook-level and function-level events for the same execution.",
    "examples": [
      "5dc6f040-0f72-4e8c-a64e-25eeef62708c"
    ]
  },
  "functionId": {
    "$id": "#/properties/functionId",
    "type": "string",
    "description": "The identifier of the function. On hook events, this is the
function configured on the hook. On function events, this is the individual function
within the executor.",
    "examples": [
      "my_first_function"
    ]
  },
  "functionType": {
    "$id": "#/properties/functionType",
    "type": "string",
    "enum": [
      "CUSTOM_OUTPUT",
      "HTTP_REQUEST",
      "SEQUENTIAL_EXECUTOR",
      "CONCURRENT_EXECUTOR"
    ]
  },

```

```
    "description": "The type of function."
  },
  "executionTimeMs": {
    "$id": "#/properties/executionTimeMs",
    "type": "integer",
    "description": "Execution time in milliseconds."
  },
  "status": {
    "$id": "#/properties/status",
    "type": "string",
    "enum": [
      "SUCCESS",
      "ERROR"
    ],
    "description": "Hook execution outcome. Present on HOOK_SUMMARY events only."
  },
  "errorType": {
    "$id": "#/properties/errorType",
    "type": "string",
    "enum": [
      "INTERNAL_ERROR",
      "SYNTAX_ERROR",
      "RESOURCE_LIMIT_ERROR",
      "RESTRICTION_ERROR",
      "TIMEOUT_ERROR",
      "VALIDATION_ERROR"
    ],
    "description": "Structured error classification. Present on error events only."
  },
  "cause": {
    "$id": "#/properties/cause",
    "type": "string",
    "description": "Error detail string describing the specific failure."
  },
  "input": {
    "$id": "#/properties/input",
    "type": "object",
    "description": "Before-values of variables targeted by output expressions."
  },
  "output": {
    "$id": "#/properties/output",
    "type": "object",
    "description": "Values produced by the function. Present on FUNCTION_COMPLETED events only."
  }
}
```

```
},
"http": {
  "$id": "#/properties/http",
  "type": "object",
  "description": "HTTP request and response details for HTTP_REQUEST functions.",
  "properties": {
    "request": {
      "type": "object",
      "description": "The HTTP request details.",
      "properties": {
        "url": { "type": "string", "description": "The request URL." },
        "method": { "type": "string", "description": "The HTTP method." },
        "headers": {
          "type": "array",
          "description": "The request headers.",
          "items": {
            "type": "object",
            "properties": {
              "name": { "type": "string" },
              "value": { "type": "string" }
            }
          }
        }
      }
    },
    "body": { "type": ["string", "null"], "description": "The request body." }
  }
},
"responseTimeMs": { "type": ["integer", "null"], "description": "HTTP response
time in milliseconds." },
"response": {
  "type": ["object", "null"],
  "description": "The HTTP response details. Null when the server was never
reached.",
  "properties": {
    "statusCode": { "type": ["integer", "null"], "description": "The HTTP
status code. Null when the server was never reached." },
    "headers": {
      "type": ["array", "null"],
      "description": "The response headers.",
      "items": {
        "type": "object",
        "properties": {
          "name": { "type": "string" },
          "value": { "type": "string" }
        }
      }
    }
  }
}
```

```

    }
  },
  "body": { "type": ["string", "null"], "description": "The response body." }
},
"truncated": { "type": "boolean", "description": "True when request or response
body was truncated to fit within log event size limits." }
}
},

```

"__COMMENT_oneOf": "The oneOf section defines subtypes for our events. Subtypes can have different rules, including which fields are required. For more information, see <https://json-schema.org/understanding-json-schema/reference/combining.html#oneof> ",

```

"oneOf": [
  { "$ref": "#/definitions/eventAdMarkersFound" },
  { "$ref": "#/definitions/eventNonAdMarkerFound" },
  { "$ref": "#/definitions/eventMakingAdsRequest" },
  { "$ref": "#/definitions/eventModifiedTargetUrl" },
  { "$ref": "#/definitions/eventRawAdsResponse" },
  { "$ref": "#/definitions/eventVastResponse" },
  { "$ref": "#/definitions/eventFilledAvail" },
  { "$ref": "#/definitions/eventFilledOverlayAvail" },
  { "$ref": "#/definitions/eventErrorFiringBeaconFailed" },
  { "$ref": "#/definitions/eventWarningNoAdvertisements" },
  { "$ref": "#/definitions/eventUnknownHost" },
  { "$ref": "#/definitions/eventErrorAdsTimeout" },
  { "$ref": "#/definitions/eventErrorVastMissingOverlays" },
  { "$ref": "#/definitions/eventPlannedAvail" },
  { "$ref": "#/definitions/eventEmptyVastResponse" },
  { "$ref": "#/definitions/eventEmptyVmapResponse" },
  { "$ref": "#/definitions/eventErrorUnknown" },
  { "$ref": "#/definitions/eventVastRedirect" },
  { "$ref": "#/definitions/eventRedirectedVastResponse" },
  { "$ref": "#/definitions/eventErrorAdsMissingImpression" },
  { "$ref": "#/definitions/eventErrorAdsResponseParse" },
  { "$ref": "#/definitions/eventErrorAdsInvalidResponse" },
  { "$ref": "#/definitions/eventErrorDisallowedHost" },
  { "$ref": "#/definitions/eventPersonalizationDisabled" },
  { "$ref": "#/definitions/eventWarningDynamicVariableSubFailed" },
  { "$ref": "#/definitions/eventVodTimeBasedAvailPlanVastResponseForOffset" },
  { "$ref": "#/definitions/eventVodTimeBasedAvailPlanSuccess" },
  { "$ref": "#/definitions/eventPreAdsRequestHookSummary" },

```

```

    { "$ref": "#/definitions/eventPreAdsRequestHookError" },
    { "$ref": "#/definitions/eventPreAdsRequestFunctionCompleted" },
    { "$ref": "#/definitions/eventPreAdsRequestFunctionError" }
  ],

  "definitions": {
    "eventAdMarkersFound": {
      "$id": "#/definitions/eventAdMarkersFound",
      "required": [
        "eventType",
        "adMarkers"
      ],
      "properties": {
        "eventType": {
          "type": "string",
          "const": "AD_MARKER_FOUND"
        }
      }
    },
    "eventNonAdMarkerFound": {
      "$id": "#/definitions/eventNonAdMarkerFound",
      "required": [
        "eventType",
        "adMarkers"
      ],
      "properties": {
        "eventType": {
          "type": "string",
          "const": "NON_AD_MARKER_FOUND"
        }
      }
    },
    "eventMakingAdsRequest": {
      "$id": "#/definitions/eventMakingAdsRequest",
      "required": [
        "eventType",
        "adsRequestUrl"
      ],
      "properties": {
        "eventType": {
          "type": "string",
          "const": "MAKING_ADS_REQUEST"
        }
      }
    }
  }
}

```

```

    },

    "eventModifiedTargetUrl": {
      "$id": "#/definitions/eventModifiedTargetUrl",
      "required": [
        "eventType",
        "originalTargetUrl",
        "updatedTargetUrl"
      ],
      "properties": {
        "eventType": {
          "type": "string",
          "const": "MODIFIED_TARGET_URL"
        }
      }
    }
  },

```

```

    "eventRawAdsResponse": {
      "$id": "#/definitions/eventRawAdsResponse",
      "required": [
        "eventType",
        "rawAdsResponse",
        "rawAdsResponseIndex"
      ],
      "properties": {
        "eventType": {
          "type": "string",
          "const": "RAW_ADS_RESPONSE"
        }
      }
    }
  },

```

```

    "eventVastResponse": {
      "$id": "#/definitions/eventVastResponse",
      "_comment": "NOTE: the vastResponse property should ideally be marked as a
required field for this event, but unfortunately, in the case of an empty vast
response, we currently emit an EMPTY_VAST_RESPONSE followed by a VAST_RESPONSE, and
the vastResponse property is not present in the latter. We need to fix this so that we
don't emit both of those events in the empty response case, and update this schema to
flag vastResponse as required for VAST_RESPONSE.",
      "required": [
        "eventType"
      ],
      "properties": {

```



```
    "eventType": {
      "type": "string",
      "const": "VAST_RESPONSE"
    }
  },
  "eventFilledAvail": {
    "$id": "#/definitions/eventFilledAvail",
    "required": [
      "eventType",
      "avail"
    ],
    "properties": {
      "eventType": {
        "type": "string",
        "const": "FILLED_AVAIL"
      }
    }
  },
  "eventFilledOverlayAvail": {
    "$id": "#/definitions/eventFilledOverlayAvail",
    "required": [
      "eventType",
      "avail"
    ],
    "properties": {
      "eventType": {
        "type": "string",
        "const": "FILLED_OVERLAY_AVAIL"
      }
    }
  },
  "eventErrorVastMissingOverlays": {
    "$id": "#/definitions/eventErrorVastMissingOverlays",
    "required": [
      "eventType",
      "adsRequestUrl",
      "requestHeaders"
    ],
    "properties": {
      "eventType": {
```

```

        "type": "string",
        "const": "ERROR_VAST_MISSING_OVERLAYS"
    }
}
},

"eventErrorFiringBeaconFailed": {
    "$id": "#/definitions/eventErrorFiringBeaconFailed",
    "required": [
        "eventType",
        "error",
        "beaconInfo"
    ],
    "properties": {
        "eventType": {
            "type": "string",
            "const": "ERROR_FIRING_BEACON_FAILED"
        }
    }
},

"eventWarningNoAdvertisements": {
    "$id": "#/definitions/eventWarningNoAdvertisements",
    "required": [
        "eventType"
    ],
    "_comment": "We should really have a more descriptive error field for these events",
    "properties": {
        "eventType": {
            "type": "string",
            "const": "WARNING_NO_ADVERTISEMENTS"
        }
    }
},

"eventUnknownHost": {
    "$id": "#/definitions/eventUnknownHost",
    "required": [
        "eventType",
        "requestHeaders"
    ],
    "properties": {
        "eventType": {

```

```

        "type": "string",
        "const": "ERROR_UNKNOWN_HOST"
    }
}
},

"eventErrorAdsTimeout": {
    "$id": "#/definitions/eventErrorAdsTimeout",
    "required": [
        "eventType",
        "adsRequestUrl",
        "requestHeaders"
    ],
    "properties": {
        "eventType": {
            "type": "string",
            "const": "ERROR_ADS_TIMEOUT"
        }
    }
},

"eventPlannedAvail": {
    "$id": "#/definitions/eventPlannedAvail",
    "required": [
        "eventType"
    ],
    "_comment": "TODO: Flesh this out as we implement it",
    "properties": {
        "eventType": {
            "type": "string",
            "const": "PLANNED_AVAIL"
        }
    }
},

"eventEmptyVastResponse": {
    "$id": "#/definitions/eventEmptyVastResponse",
    "required": [
        "eventType"
    ],
    "properties": {
        "eventType": {
            "type": "string",
            "const": "EMPTY_VAST_RESPONSE"
        }
    }
}

```

```

    }
  }
},

"eventEmptyVmapResponse": {
  "$id": "#/definitions/eventEmptyVmapResponse",
  "required": [
    "eventType"
  ],
  "properties": {
    "eventType": {
      "type": "string",
      "const": "EMPTY_VMAP_RESPONSE"
    }
  }
},

"eventErrorUnknown": {
  "$id": "#/definitions/eventErrorUnknown",
  "required": [
    "eventType"
  ],
  "_comment": "TODO: we should have a field for the exception message or
something",
  "properties": {
    "eventType": {
      "type": "string",
      "const": "ERROR_UNKNOWN"
    }
  }
},

"eventVastRedirect": {
  "$id": "#/definitions/eventVastRedirect",
  "required": [
    "eventType"
  ],
  "properties": {
    "eventType": {
      "type": "string",
      "const": "VAST_REDIRECT"
    }
  }
},

```

```
"eventRedirectedVastResponse": {
  "$id": "#/definitions/eventRedirectedVastResponse",
  "required": [
    "eventType"
  ],
  "properties": {
    "eventType": {
      "type": "string",
      "const": "REDIRECTED_VAST_RESPONSE"
    }
  },
  "_comment": "NOTE that the property vastResponse is not required because empty vast responses do not contain a vastResponse."
},

"eventErrorAdsResponseParse": {
  "$id": "#/definitions/eventErrorAdsResponseParse",
  "required": [
    "eventType"
  ],
  "_comment": "We should have a field with an error message here",
  "properties": {
    "eventType": {
      "type": "string",
      "const": "ERROR_ADS_RESPONSE_PARSE"
    }
  }
},

"eventErrorAdsInvalidResponse": {
  "$id": "#/definitions/eventErrorAdsInvalidResponse",
  "required": [
    "eventType",
    "additionalInfo"
  ],
  "properties": {
    "eventType": {
      "type": "string",
      "const": "ERROR_ADS_INVALID_RESPONSE"
    }
  }
},
```

```
"eventErrorAdsMissingImpression": {
  "$id": "#/definitions/eventErrorAdsMissingImpression",
  "required": [
    "eventType"
  ],
  "properties": {
    "eventType": {
      "type": "string",
      "const": "ERROR_VAST_MISSING_IMPRESSION"
    }
  }
},

"eventErrorDisallowedHost": {
  "$id": "#/definitions/eventErrorDisallowedHost",
  "required": [
    "eventType"
  ],
  "properties": {
    "eventType": {
      "type": "string",
      "const": "ERROR_DISALLOWED_HOST"
    }
  }
},

"eventPersonalizationDisabled": {
  "$id": "#/definitions/eventPersonalizationDisabled",
  "required": [
    "eventType"
  ],
  "properties": {
    "eventType": {
      "type": "string",
      "const": "ERROR_PERSONALIZATION_DISABLED"
    }
  }
},

"eventWarningDynamicVariableSubFailed": {
  "$id": "#/definitions/eventWarningDynamicVariableSubFailed",
  "required": [
    "eventType",
    "adsRequestUrl"
  ]
}
```

```

    ],
    "properties": {
      "eventType": {
        "type": "string",
        "const": "WARNING_URL_VARIABLE_SUBSTITUTION_FAILED"
      }
    }
  },

  "eventVodTimeBasedAvailPlanVastResponseForOffset": {
    "$id": "#/definitions/eventVodTimeBasedAvailPlanVastResponseForOffset",
    "required": [
      "eventType",
      "vastResponse"
    ],
    "properties": {
      "eventType": {
        "type": "string",
        "const": "VOD_TIME_BASED_AVAIL_PLAN_VAST_RESPONSE_FOR_OFFSET"
      }
    }
  },

  "eventVodTimeBasedAvailPlanSuccess": {
    "$id": "#/definitions/eventVodTimeBasedAvailPlanSuccess",
    "required": [
      "eventType",
      "vodCreativeOffsets"
    ],
    "properties": {
      "eventType": {
        "type": "string",
        "const": "VOD_TIME_BASED_AVAIL_PLAN_SUCCESS"
      }
    }
  },

  "creativeUniqueId": {
    "type": "string",
    "description": "The unique identifier for the ad, used as a key for transcoding. This is the ID field for the creative in the VAST response, if available. Otherwise, it's the mezzanine URL of the ad. "
  },

```

```

    "adSystem": {
      "type": "string",
      "description": "The value of the <code>AdSystem</code> tag in the VAST response."
    },

    "vastDuration": {
      "type": "number",
      "description": "The duration of the ad, as parsed from the VAST response."
    },

    "transcodedAdDuration": {
      "type": "number",
      "description": "The duration of the ad, calculated from the transcoded asset."
    },

    "adContent": {
      "$id": "#/properties/adContent",
      "type": ["object", "null"],
      "title": "adContent",
      "description": "Information about the content of the inserted ad.",
      "additionalProperties": false,
      "properties": {
        "adPlaylistUri": {
          "$id": "#/properties/adContent/adPlaylistUri",
          "type": "object",
          "title": "adPlaylistUri",
          "description": "The mapping from the origin manifest for a variant to the ad manifest for the variant. For DASH, this contains a single entry, because all variants are represented in a single DASH manifest. ",
          "additionalProperties": {
            "$id": "#/properties/adContent/adPlaylistUri/adPlaylistUri",
            "type": "string",
            "description": "The URL of the ad manifest for the specific variant."
          }
        }
      }
    },

    "adMezzanineUri": {
      "type": "string",
      "description": "The URL of the mezzanine version of the ad, which is the input to the transcoder."
    },

```



```
"bitrate": {
  "type": "integer",
  "examples": [
    533
  ],
  "description": "The bitrate."
},
"width": {
  "type": "integer",
  "examples": [
    1280
  ],
  "description": "Width in pixels."
},
"height": {
  "type": "integer",
  "examples": [
    720
  ],
  "description": "Height in pixels."
},

"trackingEvents": {
  "type": "object",
  "title": "trackingEvents",
  "description": "The tracking beacon URLs for the various tracking events for the
ad. The keys are the event names, and the values are a list of beacon URLs.",

  "additionalProperties": {
    "type": "array",
    "description": "The list of beacon URLs for the specified tracking event
(impression, complete, and so on)",
    "items": {
      "type": "string",
      "description": "The beacon URLs for this tracking event."
    }
  }
},

"nonLinearAds": {
  "$id": "#/properties/nonLinearAds",
  "type": "object",
  "title": "nonLinearAds",
```

```

    "description": "A NonLinearAds as it appears in the VAST response.",
    "required": [
        "nonLinearAdList",
        "nonLinearTrackingEvents"
    ],
    "properties": {
        "nonLinearAdList": {
            "type": "array",
            "description": "List of non linear ads as they exist within one
NonLinearAds.",
            "items": {
                "type": "object",
                "title": "nonLinearAdList",
                "description": "List of NonLinearAd as they are parsed from its parent
NonLinearAds.",
                "properties": {
                    "nonLinearAdId": {
                        "type": "string",
                        "description": "Ad ID of this non linear ad."
                    },
                    "nonLinearAdSystem": {
                        "type": "string",
                        "description": "Ad system of this non linear ad's parent Inline ad."
                    },
                    "nonLinearAdTitle": {
                        "type": "string",
                        "description": "Ad title of this non linear ad's parent Inline ad."
                    },
                    "nonLinearCreativeId": {
                        "type": "string",
                        "description": "Creative ID of this non linear ad's parent Creative
ad."
                    },
                    "nonLinearCreativeAdId": {
                        "type": "string",
                        "description": "Creative ad ID of this non linear ad."
                    },
                    "nonLinearCreativeSequence": {
                        "type": "string",
                        "description": "Creative sequence of this non linear ad."
                    },
                    "nonLinearWidth": {
                        "type": "string",
                        "description": "Width of this non linear ad."
                    }
                }
            }
        }
    }

```

```
    },
    "nonLinearHeight": {
      "type": "string",
      "description": "Height of this non linear ad."
    },
    "nonLinearExpandedWidth": {
      "type": "string",
      "description": "Expanded width of this non linear ad."
    },
    "nonLinearExpandedHeight": {
      "type": "string",
      "description": "Expanded height of this non linear ad."
    },
    "nonLinearScalable": {
      "type": "boolean",
      "description": "Boolean denoting if this non linear ad is scalable."
    },
    "nonLinearMaintainAspectRatio": {
      "type": "boolean",
      "description": "Boolean denoting if aspect ratio should be maintained
for this non linear ad."
    },
    "nonLinearMinSuggestedDuration": {
      "type": "string",
      "description": "Min suggested duration for this non linear ad."
    },
    "nonLinearApiFramework": {
      "type": "string",
      "description": "API framework for this non linear ad's parent Inline
ad."
    },
    "nonLinearStaticResource": {
      "type": "string",
      "description": "Static resource for this non linear ad."
    },
    "nonLinearStaticResourceCreativeType": {
      "type": "string",
      "description": "Static Resource creative type for this non linear ad."
    },
    "nonLinearIFrameResource": {
      "type": "string",
      "description": "I-Frame resource for this non linear ad."
    },
    "nonLinearHtmlResource": {
```

```

        "type": "string",
        "description": "HTML resource for this non linear ad."
    },
    "nonLinearAdParameters": {
        "type": "string",
        "description": "Ad parameters for this non linear ad."
    },
    "nonLinearClickThrough": {
        "type": "string",
        "description": "Click Through data for this non linear ad."
    },
    "nonLinearClickTracking": {
        "type": "string",
        "description": "Click Tracking data for this non linear ad."
    },
    "nonLinearClickTrackingId": {
        "type": "string",
        "description": "Click Tracking ID for this non linear ad."
    }
}
},
"nonLinearTrackingEvents": { "$ref": "#/definitions/trackingEvents" },
"extensions": {
    "$id": "#/properties/nonLinearAds/extensions",
    "type": "array",
    "description": "Extensions that exist for this NonLinearAds.",
    "items": {
        "$id": "#/properties/nonLinearAds/extensions/items",
        "type": "object",
        "title": "Extensions",
        "description": "Extensions found in non linear ads",
        "additionalProperties": false,
        "properties": {
            "extensionType": {
                "$id": "#/properties/nonLinearAds/extensions/extensionType",
                "type": "string",
                "description": "The value of the extension type attribute of the
<code>Extensions</code> tag.",
                "examples": [
                    "FreeWheel"
                ]
            },

```

```
        "$id": "#/properties/nonLinearAds/extensions/extensionContent",
        "type": "string",
        "description": "The extension content attribute of the
<code>Extensions</code> tag.",
        "examples": [
            "progressive"
        ]
    }
}
}
},
{
    "adBreakTrackingEvents": {
        "$id": "#/properites/adBreakTrackingEvents",
        "type": "object",
        "title": "adBreakTrackingEvents",
        "description": "These are all VMAP ad break tracking events.",
        "additionalProperties": {
            "type": "array",
            "description": "VMAP/ad break tracking events and corresponding URL",
            "items": {
                "type": "string",
                "description": "The beacon URLs for this tracking event."
            }
        }
    },
    "vastAd": {
        "$id": "#/properties/vastAd",
        "type": "object",
        "title": "vastAd",
        "description": "Information about a single ad parsed from the VAST response.",
        "required": [
            "vastAdId",
            "adSystem",
            "adTitle",
            "creativeId",
            "creativeAdId",
            "duration",
            "vastMediaFiles",
            "trackingEvents"
        ],
        "additionalProperties": false,
        "properties": {
```

```

    "vastAdId": {
      "$id": "#/properties/vastAd/vastAdId",
      "type": "string",
      "description": "The value of the id attribute of the <code>Ad</code> tag in
the VAST response",
      "examples": [
        "ad1"
      ]
    },
    "adSystem": {"$ref": "#/definitions/adSystem" } ,
    "adTitle": {
      "$id": "#/properties/vastAd/adTitle",
      "type": "string",
      "description": "The media files that are available for the ad in the VAST
response.",
      "examples": [
        "External NCA1C1L1 LinearInlineSkippable"
      ]
    },
    "creativeId": {
      "$id": "#/properties/vastAd/creativeId",
      "type": "string",
      "description": "The value of the id attribute of the <code>Creative</code>
tag in the VAST response.",
      "examples": [
        "creative1"
      ]
    },
    "creativeAdId": {
      "$id": "#/properties/vastAd/creativeAdId",
      "type": "string",
      "description": "The value of the adId attribute of the <code>Creative</code>
tag in the VAST response."
    },
    "duration": {
      "$id": "#/properties/vastAd/duration",
      "type": "number",
      "description": "The approximate duration of the ad, based on the
<code>duration</code> tag in the <code>linear</code> element of the VAST response.",
      "examples": [
        30,
        30.0
      ]
    }
  },

```

```

    "vastMediaFiles": {
      "$id": "#/properties/vastAd/vastMediaFiles",
      "type": "array",
      "description": "The list of available media files for the ad in the VAST
response.",
      "items": {
        "$id": "#/properties/vastAd/vastMediaFiles/items",
        "type": "object",
        "title": "vastMediaFile",
        "description": "Information about a media file for the ad.",
        "required": [
          "uri",
          "id",
          "delivery",
          "type",
          "apiFramework",
          "width",
          "height",
          "bitrate"
        ],
        "additionalProperties": false,
        "properties": {
          "uri": { "$ref": "#/definitions/adMezzanineUri" },
          "id": {
            "$id": "#/properties/vastAd/vastMediaFiles/items/properties/id",
            "type": "string",
            "description": "The value of the id attribute of the <code>MediaFile</
code> tag.",
            "examples": [
              "GDFP"
            ]
          },
          "delivery": {
            "$id": "#/properties/vastAd/vastMediaFiles/items/properties/delivery",
            "type": "string",
            "description": "The protocol used for the media file, set to either
progressive or streaming.",
            "examples": [
              "progressive"
            ]
          },
          "type": {
            "$id": "#/properties/vastAd/vastMediaFiles/items/properties/type",
            "type": "string",

```

```

        "description": "The MIME type of the media file, taken from the type
attribute of the <code>MediaFile</code> tag.",
        "examples": [
            "video/mp4"
        ]
    },
    "apiFramework": {
        "$id": "#/properties/vastAd/vastMediaFiles/items/properties/
apiFramework",
        "type": "string",
        "description": "The API framework needed to manage the media file.
Example: <code>VPAID</code>."
    },
    "width": {
        "$ref": "#/definitions/width"
    },
    "height": {
        "$ref": "#/definitions/height"
    },
    "bitrate": {
        "$ref": "#/definitions/bitrate"
    }
}
},
"trackingEvents": { "$ref": "#/definitions/trackingEvents" },
"vastAdTagUri": {
    "$id": "#/properties/vastAd/vastAdTagUri",
    "type": "string",
    "description": "The VMAP-specific redirect URI for an ad.",
    "examples": [
        "https://ads.redirect.com/redirect1"
    ]
}
},
"eventPreAdsRequestHookSummary": {
    "$id": "#/definitions/eventPreAdsRequestHookSummary",
    "required": [
        "eventType",
        "eventId",
        "functionId",
        "executionTimeMs",

```



```
    "status"
  ],
  "properties": {
    "eventType": {
      "type": "string",
      "const": "PRE_ADS_REQUEST_HOOK_SUMMARY"
    }
  }
},

"eventPreAdsRequestHookError": {
  "$id": "#/definitions/eventPreAdsRequestHookError",
  "required": [
    "eventType",
    "eventId",
    "functionId",
    "executionTimeMs",
    "errorType",
    "cause"
  ],
  "properties": {
    "eventType": {
      "type": "string",
      "const": "PRE_ADS_REQUEST_HOOK_ERROR"
    }
  }
},

"eventPreAdsRequestFunctionCompleted": {
  "$id": "#/definitions/eventPreAdsRequestFunctionCompleted",
  "required": [
    "eventType",
    "eventId",
    "functionId",
    "functionType",
    "executionTimeMs"
  ],
  "properties": {
    "eventType": {
      "type": "string",
      "const": "PRE_ADS_REQUEST_FUNCTION_COMPLETED"
    }
  }
},
```

```
"eventPreAdsRequestFunctionError": {
  "$id": "#/definitions/eventPreAdsRequestFunctionError",
  "required": [
    "eventType",
    "eventId",
    "functionId",
    "functionType",
    "executionTimeMs",
    "errorType",
    "cause"
  ],
  "properties": {
    "eventType": {
      "type": "string",
      "const": "PRE_ADS_REQUEST_FUNCTION_ERROR"
    }
  }
}
```

AWS Elemental MediaTailor manifest logs description and event types

The following sections describe the logs that MediaTailor emits to describe events with the origin server when requesting and receiving a manifest. These are ManifestService logs.

Topics

- [ManifestService events](#)
- [Manifest logs properties](#)

ManifestService events

The following events are emitted during MediaTailor interactions with the origin.

Log	Description
CONFIG_SECURITY_ERROR	The MediaTailor configuration has a security issue.

Log	Description
CONFIG_SYNTAX_ERROR	The origin and asset path result in a malformed URL.
CONNECTION_ERROR	The MediaTailor connection to the origin was refused or failed.
GENERATED_MANIFEST	MediaTailor generated a manifest. You must have debug mode enabled to receive these logs. For information about debug log mode, including how to enable it, see Generating debug logs.
HOST_DISALLOWED	MediaTailor does not allow HTTP requests to this host.
INCOMPATIBLE_HLS_VERSION	The manifest uses an incompatible HLS version. MediaTailor requires version 3 or greater.
INVALID_SINGLE_PERIOD_DASH_MANIFEST	The single-period DASH manifest is invalid. MediaTailor is passing through single-period DASH manifest.
IO_ERROR	MediaTailor encountered an IO error during communication with the origin.
LAST_PERIOD_MISSING_AUDIO	The last period in the DASH manifest is missing all audio AdaptationSets because of origin audio or video misalignment. To avoid playback issues, delay publishing the last period until at least the next request.

Log	Description
LAST_PERIOD_MISSING_AUDIO_WARNING	The last period in the DASH manifest is missing all audio AdaptationSets because of origin audio or video misalignment. Choosing to publish (not delay) the last period. Missing audio might cause playback issues.
MANIFEST_ERROR	The MediaTailor manifest request failed.
NO_MASTER_OR_MEDIA_PLAYLIST	The origin response doesn't contain a primary playlist or media playlist.
NO_MASTER_PLAYLIST	The origin response doesn't contain the expected primary playlist.
NO_MEDIA_PLAYLIST	The origin response doesn't contain the expected media playlist.
ORIGIN_MANIFEST	MediaTailor fetched an origin manifest. You must have debug mode enabled to receive these logs. For information about debug log mode, including how to enable it, see Generating debug logs .
PARSING_ERROR	The origin is unable to parse the manifest request.
PRE_SESSION_INIT_FUNCTION_COMPLETED	An individual function in the pre-session initialization hook completed. This is an opt-in event type.
PRE_SESSION_INIT_FUNCTION_ERROR	An individual function in the pre-session initialization hook failed.
PRE_SESSION_INIT_HOOK_ERROR	The pre-session initialization hook execution failed.

Log	Description
PRE_SESSION_INIT_HOOK_SUMMARY	Summary of the pre-session initialization hook execution, including success or error status. This is an opt-in event type.
SCTE35_PARSING_ERROR	MediaTailor is unable to parse Signal Binary element in the manifest.
SESSION_INITIALIZED	A session was initialized. You must have debug mode enabled to receive these logs. For information about debug log mode, including how to enable it, see Generating debug logs.
TIMEOUT_ERROR	The MediaTailor manifest request timed out.
TRACKING_RESPONSE	MediaTailor served a tracking response. You must have debug mode enabled to receive these logs. For information about debug log mode, including how to enable it, see Generating debug logs.
UNKNOWN_ERROR	MediaTailor encountered an unknown error.
UNKNOWN_HOST	The host is unknown.
UNSUPPORTED_SINGLE_PERIOD_DASH_MANIFEST	The single-period DASH manifest is unsupported. MediaTailor is passing through single-period DASH manifest.

Manifest logs properties

This section describes the properties of the manifest logs.

Property	Type	Required
awsAccountId	string	true
eventTimestamp	string	true
originId	string	true
customerId	string	false
eventType	string	false
sessionId	string	false
originRequestUrl	string	false
mediaTailorPath	string	false
requestId	string	false
responseBody	string	false
sessionType	string (legal values: [DASH, HLS])	false
requestNextToken	string	false
eventDescription	string	false
errorType	string	false
eventId	string	false
executionTimeMs	integer	false
functionId	string	false
functionType	string	false
http	object	false

Property	Type	Required
input	object	false
output	object	false
status	string	false
assetPath	string	false
originFullUrl	string	false
originPrefixUrl	string	false
additionalInfo	string	false
cause	string	false
response	string	false
httpCode	string	false
errorMessage	string	false
adAdsResponse	string	false
adAdsRawResponse	string	false
adAdsRequest	string	false
adNumNewAvails	string	false
generatedMediaPlay list	string	false
requestStartTime	string	false
requestEndTime	string	false
requestStartTimeEp ochMillis	string	false

Property	Type	Required
requestEndTimeEpochMillis	string	false

AWS Elemental MediaTailor transcode logs description and event types

The following sections describe the logs that MediaTailor emits to describe events with the transcode service when preparing creatives for ad stitching. These are TranscodeService logs.

Topics

- [TranscodeService events](#)
- [Transcode logs properties](#)

TranscodeService events

The following events are emitted during MediaTailor interactions while transcoding ads.

Log	Description
IMPORT_ERROR	MediaTailor encountered an internal error during an import job (for preconditioned ads). Using an empty set of ads.
INITIALIZED	MediaTailor initialized either a transcode job or an import job (for preconditioned ads).
INTERNAL_ERROR	MediaTailor encountered an internal error. Using an empty set of ads.
MISSING_VARIANTS	MediaTailor could not transcode the ad because of missing variants. Using an empty set of ads.
PROFILE_NOT_FOUND	MediaTailor could not transcode the ad because of a missing profile to transcode. Using an empty set of ads.

Log	Description
TRANSCODE_COMPLETED	Video transcoding is complete. The ad can be used for ad insertion.
TRANSCODE_ERROR	MediaTailor encountered an internal error during a transcode job. Using an empty set of ads.
TRANSCODE_IN_PROGRESS	Video transcoding is in progress. The transcoded video is not ready. Using an empty set of ads.

Transcode logs properties

This section describes the properties of the transcode logs.

Property	Type	Required	Description
awsAccountId	string	true	The AWS account ID for the MediaTailor configuration that was used for the session.
eventTimestamp	string	true	The date and time of the event.
originId	string	true	The configuration name from the MediaTailor configuration. This is different from the video content source, which is also part of the configuration.

Property	Type	Required	Description
eventType	string	false	The code for the event that triggered this log message. Example: TRANSCODE_ERROR .
eventDescription	string	false	A short description of the event that triggered this log message, provided by the MediaTailor service. By default, this is empty.
sessionId	string	false	The unique numeric identifier that MediaTailor assigned to the player session. All requests that a player makes for a session have the same session ID. Example: e039fd39-09f0-46b2-aca9-9871cc116cde .
creativeUniqueId	string	false	The unique identifier for the ad creative that's being transcoded.
profileName	string	false	
adUri	string	false	The URI for the ad creative.

Property	Type	Required	Description
transcode RequestId	string	false	The unique identifier for this transcode request.
cacheStatus	string	false	Indicates if MediaTailor cached the transcoded ad.

Using vended logs to send AWS Elemental MediaTailor logs

You can use vended logs for greater flexibility and control over where to deliver logs that MediaTailor emits from your playback configuration.

With vended logs, MediaTailor sends all log activity associated with a configuration to Amazon CloudWatch Logs. CloudWatch Logs then sends the percent of logs that you specify to your chosen destination. Supported destinations are an Amazon CloudWatch Logs log group, Amazon S3 bucket, or Amazon Data Firehose stream.

Because vended logs are available at volume discount pricing, you could see cost savings compared to sending logs directly to CloudWatch Logs. For pricing, see *Vended Logs* on the **Logs** tab at [Amazon CloudWatch Pricing](#).

To use vended logs, you must do the following:

1. [Add permissions](#).
2. [Create log delivery destinations](#).
3. [Configure log delivery in CloudWatch Logs](#).
4. [Enable vended logs in MediaTailor](#).

For more information about vended logs, see [Enable logging from AWS services](#) in the CloudWatch Logs user guide. MediaTailor supports V2 of vended logs.

Step 1: Add permissions for MediaTailor log delivery

The person who's setting up vended logs must have permissions to create the delivery destination, configure log delivery, and enable vended logs in MediaTailor. Use the following policies to ensure that you have the appropriate permissions to set up vended logs.

Policies for CloudWatch Logs and delivery destinations

The following sections in the *Amazon CloudWatch Logs User Guide* provide the policies that enable you to work with logs in CloudWatch Logs and your delivery destinations. If you send logs to multiple locations, you can combine the policy statements into one policy instead of creating multiple policies.

- [Logs sent to CloudWatch Logs](#)
- [Logs sent to Amazon S3](#)
- [Logs sent to Firehose](#)

Policy for set up from the console

If you're setting up vended logs delivery through the console instead of the API or AWS CLI, you must have the following additional permissions in your policy.

Policy for vended logs in MediaTailor

To create, view, or modify vended logs delivery in MediaTailor, you must have the following permissions in your policy.

For information about adding permissions and working with policies, see [Identity and Access Management for AWS Elemental MediaTailor](#).

Step 2: Create delivery destinations for MediaTailor logs

Create the resources where your logs will be sent. Record the ARN of the resource for use in configuring the log delivery in a later step.

CloudWatch Logs log group delivery destination

Use one of the following for help creating a log group.

- For the console, see [Create a log group in CloudWatch Logs](#) in the *Amazon CloudWatch Logs User Guide*.

- For the API, see [CreateLogGroup](#) in the *Amazon CloudWatch Logs API Reference*.
- For SDKs and CLI, see [Use CreateLogGroup with an AWS SDK or AWS CLI](#) in the *Amazon CloudWatch Logs User Guide*.

Amazon S3 bucket delivery destination

Use one of the following for help creating a bucket.

- For the console, SDKs, and CLI, see [Create a bucket](#) in the *Amazon Simple Storage Service User Guide*.
- For the API, see [CreateBucket](#) in the *Amazon Simple Storage Service API Reference*.

Firehose stream delivery destination

For help creating a stream, see [Create a Firehose stream from console](#) in the *Amazon Data Firehose Developer Guide*.

Step 3: Enable vended logs for the MediaTailor playback configuration

Create or update the playback configuration that will be sending logs to the delivery destination that you created in the previous step. Record the name of the configuration for use in configuring the log delivery in a later step.

- To enable vended logs through the console, using [Creating a configuration](#) or [Editing configuration settings](#) Editing a configuration to access the **Logging** settings. For **Logging strategies**, choose **Vended logs**.
- To enable vended logs through the API, you must have an existing configuration. Use `ConfigureLogsForPlaybackConfiguration` to add the logging strategy Vended logs.

If you're using the legacy MediaTailor logging strategy of sending logs directly to CloudWatch Logs and want to migrate to vended logs, see [Migrating the logging strategy](#).

Important

If you change the log strategy from Legacy CloudWatch to vended logs, MediaTailor will make this change as soon as you save the updates. You will stop receiving logs until you have fully configured vended logging.

Step 4: Configure log delivery in CloudWatch Logs

In CloudWatch Logs, you must create three elements to represent the pieces of log delivery. These elements are described in detail in [CreateDelivery](#) in the *Amazon CloudWatch Logs API Reference*. The high-level steps to configure the delivery with the CloudWatch Logs API are as follows.

To configure log delivery in CloudWatch Logs (API)

1. Use [PutDeliverySource](#) to add the source of logs.

A `DeliverySource` represents the playback configuration that's generating the logs. You need the name of the playback configuration to create the `DeliverySource`.

2. Use [PutDeliveryDestination](#) to add the destination where logs will be written.

A `DeliveryDestination` represents the delivery destination. You need the ARN of the log group, bucket, or stream to create the `DeliveryDestination`.

3. Use [PutDeliveryDestinationPolicy](#) if you are delivering logs across accounts.

If the delivery destination is in a different account from the playback configuration, you need a `DeliveryDestinationPolicy`. This policy allows CloudWatch Logs to deliver logs to the `DeliveryDestination`.

4. Use [CreateDelivery](#) to link the `DeliverySource` to the `DeliveryDestination`.

A `Delivery` represents the connection between the `DeliverySource` and `DeliveryDestination`.

Migrating your AWS Elemental MediaTailor logging strategy

If you change the log strategy from Legacy CloudWatch to vended logs, MediaTailor will make this change as soon as you save the updates. To avoid interruptions in your logging workflow, use the following steps to migrate your logging strategy.

1. Follow the steps as described in [Using vended logs](#). For [Enable vended logs in MediaTailor](#), select *both* logging strategies (**Vended logs** and **Legacy CloudWatch**).

MediaTailor will send logs through both vended logs and directly to CloudWatch Logs.

2. Make the necessary changes in your workflow that are dependent on your logging strategy and delivery destination.

3. Revisit [Enable vended logs in MediaTailor](#) and remove **Legacy CloudWatch** from the **Logging strategies**.

Writing AWS Elemental MediaTailor logs directly to Amazon CloudWatch Logs

MediaTailor produces logs that contain detailed information about session activity and ad decision server interactions, and writes them to Amazon CloudWatch. The logs provide a sequential description of activity that occurs during the session.

MediaTailor can also use vended logs for flexibility in log delivery and volume discount pricing. For information about vended logs, see [Using vended logs](#).

Topics

- [Permissions for Amazon CloudWatch Logs](#)
- ["As Run" log for AWS Elemental MediaTailor Channel Assembly](#)
- [AWS Elemental MediaTailor ADS log analysis in Amazon CloudWatch Logs Insights](#)

Permissions for Amazon CloudWatch Logs

Use AWS Identity and Access Management (IAM) to create a role that gives AWS Elemental MediaTailor access to Amazon CloudWatch. You must perform these steps for CloudWatch Logs to be published for your account. CloudWatch automatically publishes metrics for your account.

To allow MediaTailor access to CloudWatch

1. Open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation pane of the IAM console, choose **Roles**, and then choose **Create role**.
3. Choose the **Another AWS account** role type.
4. For **Account ID**, enter your AWS account ID.
5. Select **Require external ID** and enter **Midas**. This option automatically adds a condition to the trust policy that allows the service to assume the role only if the request includes the correct `sts:ExternalId`.
6. Choose **Next: Permissions**.
7. Add a permissions policy that specifies what actions this role can complete. Select from one of the following options, and then choose **Next: Review**:

- **CloudWatchLogsFullAccess** to provide full access to Amazon CloudWatch Logs
 - **CloudWatchFullAccess** to provide full access to Amazon CloudWatch
8. For **Role name**, enter **MediaTailorLogger**, and then choose **Create role**.
 9. On the **Roles** page, choose the role that you just created.
 10. To update the principal, edit the trust relationship:
 1. On the role's **Summary** page, choose the **Trust relationship** tab.
 2. Choose **Edit trust relationship**.
 3. In the policy document, change the principal to the MediaTailor service. It should look like this:

```
"Principal": {  
    "Service": "mediatailor.amazonaws.com"  
},
```

The entire policy should read as follows:

4. Choose **Update Trust Policy**.

"As Run" log for AWS Elemental MediaTailor Channel Assembly

The *As Run* log, in the CloudWatch MediaTailor/Channel/AsRunLog log group, shows information about programs and ad breaks as they play.

When you create a channel, the As Run log is disabled by default. Using the Console or the AWS Command Line Interface (AWS CLI), you can enable and disable the As Run log state for each channel in your account.

When you enable the As Run log, MediaTailor automatically creates a service-linked role that allows MediaTailor to write and manage the As Run log in your CloudWatch Logs account. For more information about service-linked roles, see [Using service-linked roles for MediaTailor](#).

Note

The As Run Log currently only supports the default program. For now it doesn't support the alternateMedia created by program rules. This means that it currently does not generate the As Run Log for alternateMedia.

Topics

- [Enabling the As Run log](#)
- [Disabling the As Run log](#)

Enabling the As Run log

To enable the As Run log, specify the channel name and enable the *As Run* log type for that channel.

Console

To enable the As Run log when creating a channel

1. Sign in to the AWS Management Console and open the MediaTailor console at <https://console.aws.amazon.com/mediatailor/>.
2. In the navigation pane, choose **Channel assembly** > **Channels**.
3. On the navigation bar, choose **Create channel**.
4. In the **Set channel details**, **Configure outputs**, and **Access control** panes, configure your channel as desired.
5. In the **Access control** pane, choose **Next**.
6. In the **Logging** pane, under **Log types**, select **Enable as run** to enable the As Run log.

To enable the As Run log when updating a channel

Note

If the channel is currently running, you must first stop that channel before you can update it. After you stop the channel, you can choose **Actions** > **Edit** to begin updating the channel.

1. Sign in to the AWS Management Console and open the MediaTailor console at <https://console.aws.amazon.com/mediatailor/>.
2. In the navigation pane, choose **Channel assembly** > **Channels**.
3. Choose the channel that you want to update to enable the As Run log for.
4. Choose **Actions** > **Edit**.

5. In the **Set channel details**, **Configure outputs**, and **Access control** panes, update your channel configuration as desired.
6. In the **Access control** pane, choose **Next**.
7. In the **Logging** pane, under **Log types**, select **Enable as run** to enable the As Run log.

To enable the As Run log from the Logging tab

Note

If the channel is currently running, you must use the **Logging** tab instead of choosing **Actions > Edit** to enable the As Run log.

1. Sign in to the AWS Management Console and open the MediaTailor console at <https://console.aws.amazon.com/mediatailor/>.
2. In the navigation pane, choose **Channel assembly > Channels**.
3. Choose the channel that you want to enable the As Run log for.
4. In the navigation bar under the channel's name, choose **Logging**.
5. Under **Logging > Log types**, select **As run** to enable the As Run log.

AWS Command Line Interface (AWS CLI)

To enable the As Run log

Run the [configure-logs-for-channel](#) command and specify the appropriate values for the required parameters.

This example is formatted for Linux, macOS, or Unix, and it uses the backslash (\) line-continuation character to improve readability.

```
$ aws mediatailor configure-logs-for-channel \  
--channel-name MyChannel1 \  
--log-types AS_RUN
```

This example is formatted for Microsoft Windows, and it uses the caret (^) line-continuation character to improve readability.

```
C:\> aws mediatailor configure-logs-for-channel ^  
--channel-name MyChannel ^  
--log-types AS_RUN
```

Where:

- *MyChannel* is the name of the channel that you own and want to enable the As Run log for.

If the command runs successfully, you receive output similar to the following.

```
{  
  "ChannelName": "MyChannel",  
  "LogTypes": [  
    "AS_RUN"  
  ]  
}
```

Disabling the As Run log

To disable the As Run log for a channel that has it enabled, specify the channel name and disable the *As Run* log type for that channel.

Console

To disable the As Run log when updating a channel

Note

If the channel is currently running, you must first stop that channel before you can update it. After you stop the channel, you can choose **Actions** > **Edit** to begin updating the channel.

1. Sign in to the AWS Management Console and open the MediaTailor console at <https://console.aws.amazon.com/mediatailor/>.
2. In the navigation pane, choose **Channel assembly** > **Channels**.
3. Choose the channel that you want to update to enable the As Run log for.
4. Choose **Actions** > **Edit**.

5. In the **Set channel details**, **Configure outputs**, and **Access control** panes, update your channel configuration as desired.
6. In the **Access control** pane, choose **Next**.
7. In the **Logging** pane, under **Log types**, clear **Enable as run** to disable the As Run log.

To disable the As Run log from the Logging tab

Note

If the channel is currently running, you must use the **Logging** tab instead of choosing **Actions > Edit** to disable the As Run log.

1. Sign in to the AWS Management Console and open the MediaTailor console at <https://console.aws.amazon.com/mediatailor/>.
2. In the navigation pane, choose **Channel assembly > Channels**.
3. Choose the channel that you want to disable the As Run log for.
4. In the navigation bar under the channel's name, choose **Logging**.
5. Under **Logging > Log types**, clear **As run** to disable the As Run log.

AWS Command Line Interface (AWS CLI)

To disable the As Run log

Run the [configure-logs-for-channel](#) command and specify the appropriate values for the required parameters.

This example is formatted for Linux, macOS, or Unix, and it uses the backslash (\) line-continuation character to improve readability.

```
$ aws mediatailor configure-logs-for-channel \  
--channel-name MyChannel \  
--log-types
```

This example is formatted for Microsoft Windows, and it uses the caret (^) line-continuation character to improve readability.

```
C:\> aws mediatailor configure-logs-for-channel ^  
--channel-name MyChannel ^  
--log-types
```

Where:

- *MyChannel* is the name of the channel that you own and want to disable the As Run log for.

If the command runs successfully, you receive output similar to the following.

```
{  
  "ChannelName": "MyChannel",  
  "LogTypes": []  
}
```

AWS Elemental MediaTailor ADS log analysis in Amazon CloudWatch Logs Insights

You can view and query AWS Elemental MediaTailor ad decision server (ADS) logs using Amazon CloudWatch Logs Insights. MediaTailor sends event logs to CloudWatch for normal processing and error conditions. The logs adhere to a JSON schema. Through CloudWatch Logs Insights, you can select logs by time frame, and then run queries against them.

For general information, see [Analyze log data with CloudWatch Logs insights](#).

Note

To access the logs, you need permissions to access Amazon CloudWatch. For instructions, see [Permissions for Amazon CloudWatch Logs](#).

To view and query ADS logs using the CloudWatch console

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, under **Logs**, choose **Insights**.
3. In the search bar, enter **AdDec**, and then from the drop-down list, select MediaTailor/AdDecisionServerInteractions.

4. (Optional) Adjust the time period that you want to study.
5. (Optional) Change the query in the dialog box. For general guidance, see [CloudWatch Logs insights query syntax](#). For examples of queries for MediaTailor ADS, see [Querying the ADS logs](#).
6. Choose **Run query**. The query might take a few seconds, during which time **Cancel** appears in place of **Run query**.
7. (Optional) To export the results as a CSV file, choose **Actions**, and then choose **Download query results (CSV)**.

Note

The console limits the number of records that it returns in query results and that it exports, so for bulk data, use the API, the AWS Command Line Interface (AWS CLI), or an SDK.

Topics

- [Querying the ADS logs](#)

Querying the ADS logs

CloudWatch Logs Insights provides a rich set of options for querying your logs. For detailed information about querying syntax, see [CloudWatch Logs insights query syntax](#). This section provides examples of common queries to get you started with your ADS logs queries. All queries run against the logs for the current time range setting.

The following query retrieves all information from the ADS logs.

```
fields @timestamp, eventType, sessionId, requestId, @message
| sort sessionId, @timestamp asc
```

The following query retrieves all requests to the ADS. This query shows a way to retrieve the request header contents for MediaTailor logs.

```
fields @timestamp, adsRequestUrl, requestHeaders.0.value as @userAgent,
requestHeaders.1.value as @xForwardedFor, sessionId, requestId
| filter eventType = "MAKING_ADS_REQUEST"
| sort @timestamp asc
```

The following query retrieves the ads MediaTailor inserted for a given session.

```
fields @timestamp, sessionId, requestId, @message
| filter eventType = "FILLED_AVAIL"
| sort @timestamp asc
```

The following query retrieves the tracking URLs that MediaTailor called on behalf of the player.

```
fields @timestamp, beaconInfo.trackingEvent, beaconInfo.beaconUri,
  beaconInfo.headers.0.value as @userAgent, beaconInfo.headers.1.value as
  @xForwardedFor, sessionId, requestId
| filter eventType = "BEACON_FIRED"
| sort @timestamp asc
```

The following query retrieves information for a specific playback session, by filtering the results by sessionId.

```
fields @timestamp, eventType, sessionId, requestId, @message
| filter sessionId = "0aaf6507-c6f9-4884-bfe7-f2f841cb8195"
| sort @timestamp asc
```

The following query retrieves information for a single request, by filtering the results by requestId.

```
fields @timestamp, eventType, sessionId, requestId, @message
| filter requestId = "f5d3cf39-6258-4cf1-b3f6-a34ff8bf641d"
| sort @timestamp asc
```

The following query retrieves a count of log entries for each event type that was logged.

```
fields eventType
| stats count() as @eventCount by eventType
```

The following query retrieves the avail ID and list of skipped ads for all avails that had skipped ads.

```
fields avail.availId
| parse @message '"skippedAds":[*]' as @skippedAdsList
| filter ispresent(@skippedAdsList)
```

Controlling the volume of AWS Elemental MediaTailor logs

MediaTailor ad insertion session logs are sometimes verbose. To reduce log costs, you can define the percentage of session logs that MediaTailor sends to Amazon CloudWatch Logs. For example, if your playback configuration has 1000 ad insertion sessions and you set a percentage enabled value of 60, MediaTailor sends logs for 600 of the sessions to CloudWatch Logs. MediaTailor decides at random which of the sessions to send logs for. If you want to view logs for a specific session, you can use the [debug log mode](#).

When you set a logging percentage, MediaTailor automatically creates a service-linked role that grants MediaTailor the permissions it requires to write CloudWatch Logs to your account. For information about how MediaTailor uses service-linked roles, see [Using service-linked roles for MediaTailor](#).

Creating a log configuration

To control the percentage of session logs that MediaTailor writes to CloudWatch Logs, you create a *log configuration* for your playback configuration. When you create a log configuration, you specify a *playback configuration name*, and a *percent enabled* value.

Console

To create a log configuration for an *existing* playback configuration

1. Sign in to the AWS Management Console and open the MediaTailor console at <https://console.aws.amazon.com/mediatailor/>.
2. On the **Playback configuration** pane, select the playback configuration that you'd like to set the log configuration for.
3. Choose **Edit**.
4. Under **Log configuration**, specify a **percent enabled** value.

To create a log configuration for a *new* playback configuration

- Follow the procedure in [Log configuration](#).

AWS Command Line Interface (AWS CLI)

To create a log configuration for an *existing* playback configuration

To create a log configuration by using the AWS CLI, run the [configure-logs-for-playback-configuration](#) command and specify the appropriate values for the required parameters.

This example is formatted for Linux, macOS, or Unix, and it uses the backslash (\) line-continuation character to improve readability.

```
$ aws mediatailor configure-logs-for-playback-configuration \  
--percent-enabled 10 \  
--playback-configuration-name MyPlaybackConfiguration
```

This example is formatted for Microsoft Windows, and it uses the caret (^) line-continuation character to improve readability.

```
C:\> aws mediatailor configure-logs-for-playback-configuration ^  
--percent-enabled 10 ^  
--playback-configuration-name MyPlaybackConfiguration
```

Where:

- *percent-enabled* is the percentage of playback configuration session logs that MediaTailor sends to CloudWatch Logs.
- *playback-configuration-name* is the name of the playback configuration to set the log configuration settings for.

If the command runs successfully, you receive output similar to the following.

```
{  
  "PercentEnabled": 10,  
  "PlaybackConfigurationName": "MyPlaybackConfiguration"  
}
```

To create a log configuration for a *new* playback configuration

- Use the `configure-logs-for-playback-configuration` option for the [put-playback-configuration](#) command.

Deactivating a log configuration

After you create a log configuration, you can't delete it—you can only *deactivate* it. To deactivate the log configuration, set the **percent enabled** value to **0** with the MediaTailor console or API. This turns off all session logging for that playback configuration.

If you want to delete the service-linked role that MediaTailor uses for the log configuration(s) in your account, you must first deactivate all of your log configurations. For information about how to delete the service-linked role, see [Using service-linked roles for MediaTailor](#).

Console

To deactivate log configuration on a playback configuration

1. Sign in to the AWS Management Console and open the MediaTailor console at <https://console.aws.amazon.com/mediatailor/>.
2. On the **Playback configuration** pane, select the playback configuration that you'd like to deactivate log configuration on.
3. Choose **Edit**.
4. Under **Log configuration**, set the **percent enabled** value to 0. This turns off all session logging for this playback configuration.
5. Select **Save**.

AWS Command Line Interface (AWS CLI)

To deactivate a log configuration

- Set the percent-enabled value to 0 using the [configure-logs-for-playback-configuration](#) command.

Filtering AWS Elemental MediaTailor logs and events

Logs emitted from a playback configuration in MediaTailor include information about a variety of activities that happen during the playback session. These activities are identified in the event type of the logs. Many events are logged by default. To help control the cost of logs in Amazon CloudWatch, you can specify the logs that MediaTailor emits.

MediaTailor provides you control over log filtering so you can do the following:

- Specify the log events that you want to exclude from logs
- Enable logging raw responses from the ad decision server (ADS)

You can set these log filtering preferences independently for each playback session, or as a default for all playback sessions for a playback configuration.

- To filter logs on a per-session basis, include query parameters in the playback session initialization request.
- To filter logs on a per-playback configuration basis, use the MediaTailor console or API to indicate your preferences in the playback configuration settings.

The following sections provide instruction for enabling log filtering on sessions and playback configurations.

Per-session log filters

To define a customized level of log detail for each session, append the following parameters to your initial server-side or client-side playback session request. Add values to the parameters to represent the events that you want to include or exclude, in a comma-delimited format:

- `aws.adsInteractionLogPublishOptInEventTypes` to receive logs for specific ad decision server (ADS) interactions.
- `aws.adsInteractionLogExcludeEventTypes` to stop receiving logs for specific ADS interactions.
- `aws.manifestServiceLogExcludeEventTypes` to stop receiving logs for specific manifest service interactions.

For a list of log and event types that MediaTailor emits, see [Manifest logs](#), [ADS logs](#), and [Transcode logs](#).

If you don't pass through any query parameters for log filtering, MediaTailor writes all logs to your delivery destination.

Example server-side session initialization with log filters

To exclude `GENERATED_MANIFEST` and `PARSING_ERROR` events from your manifest logs and `MAKING_ADS_REQUEST` from the ADS logs, the session initialization request would look like this:

```
GET <mediatailorURL>/v1/master/<hashed-account-id>/<origin-id>/index.m3u8?
aws.logMode=DEBUG&aws.manifestServiceLogExcludeEventTypes=GENERATED_MANIFEST,PARSING_ERROR&aws.
```

To enable raw logs from your ADS, include the RAW_ADS_RESPONSE value for the `AdsInteractionPublishOptInEventType` parameter:

```
GET <mediatailorURL>/v1/master/<hashed-account-id>/<origin-id>/index.m3u8?
aws.adsInteractionPublishOptInEventType=RAW_ADS_RESPONSE
```

Example client-side session initialization with log filters

To exclude log events during client-side session initialization, include `availSuppression` and log-type parameters in your client's POST request to MediaTailor. For more information about how to construct a client-side playback session request, see [Client-side ad tracking](#). The following example excludes `CONFIG_SECURITY_ERROR` and `PARSING_ERROR` events from your manifest logs and `MAKING_ADS_REQUEST` from the ADS logs.

```
POST parent.m3u8
{
  "adsInteractionLog": {
    ...
    "excludeEventTypes": [
      "MAKING_ADS_REQUEST"
    ]
  },
  "manifestServiceLog": {
    ...
    "excludeEventTypes": [
      "GENERATED_MANIFEST",
      "PARSING_ERROR"
    ]
  },
  "logMode": "DEBUG"
}
```

To enable raw logs from your ADS, include the RAW_ADS_RESPONSE value for the `publishOptInEventTypes` parameter:

```
POST parent.m3u8
{
```

```
"adsInteractionLog": {
  "publishOptInEventTypes": ["RAW_ADS_RESPONSE"],
  "excludeEventTypes": [
    "MAKING_ADS_REQUEST"
  ]
},
"manifestServiceLog": {
  ...
  "excludeEventTypes": [
    "GENERATED_MANIFEST",
    "PARSING_ERROR"
  ]
},
"logMode": "DEBUG"
}
```

Per-playback configuration log filters

Use the playback configuration's settings to define the log event types that MediaTailor emits as a default from this playback configuration. MediaTailor uses these default log filtering settings for all sessions that don't include filtering query parameters on the session init request.

You can choose to do the following:

- Receive logs for specific ad decision server (ADS) interactions.
- Exclude logs for specific ADS interactions.
- Exclude logs for specific manifest service interactions.

To set these settings from the MediaTailor console, see [Creating a configuration](#). For the MediaTailor API, see [PutPlaybackConfiguration](#) in the *AWS Elemental MediaTailor API Reference*.

For a list of log and event types that MediaTailor emits, see [Manifest logs](#), [ADS logs](#), and [Transcode logs](#).

Generating AWS Elemental MediaTailor debug logs

Use debug logs to troubleshoot MediaTailor ad insertion playback session issues. To generate debug logs, set the log mode to debug in the player's request to MediaTailor. For server-side reporting, set the log mode in the *playback request*. For client-side reporting, set the log mode in the *session initialization request*.

When the log mode is set to debug, MediaTailor writes all log event types to CloudWatch Logs. The logs provide information about the following events. For a complete list of the data produced in the debug logs, see [Debug log fields](#).

- **Origin interaction** – Details about MediaTailor interactions with the origin server. For example, the origin manifest response, manifest type, and origin URL.
- **Generated manifest** – Details about the playback session response from MediaTailor. For example, the manifest that MediaTailor generates.
- **Session initialized** – Session initialization details, such as the session ID.

To customize the log event types that you receive on a per-session basis, see [Filtering logs and events](#).

Prerequisites

To set the log mode to debug, first you need to grant MediaTailor permission to send logs to CloudWatch, if you haven't already. Once you've granted permission for MediaTailor to access CloudWatch, then you're ready to enable the debug log mode. For information about how to grant MediaTailor permission to access CloudWatch see [Setting Up Permissions for Amazon CloudWatch](#).

How to set the log mode to debug

This section explains how to set the log mode to debug for server-side reporting and client-side reporting.

Server-side reporting

For server-side reporting, include the `?aws.LogMode=DEBUG` query parameter and value in your player's GET HTTP playback request to the HLS or DASH MediaTailor endpoint. For general information about server-side reporting see [Server-side Reporting](#).

Important

The DEBUG value is case-sensitive.

A playback request that includes `?aws.LogMode=DEBUG` looks like the following:

Example Playback request to an HLS endpoint

```
GET <mediatailorURL>/v1/master/<hashed-account-id>/<origin-id>/<asset-id>?  
aws.logMode=DEBUG
```

After you set the log mode to debug, we recommend that you verify that the debug logging session is active. To verify that the debug session is active, check to see if there are any CloudWatch logs for the session ID. The session ID is included in the playback endpoint that MediaTailor provides. For more information, see [Verify that the debug log mode is active for your playback session](#).

Client-side reporting

For client-side reporting, include the `logMode` key and `DEBUG` value in your client's POST HTTP session initialization request body to the MediaTailor `/v1/session` endpoint. For general information about client-side reporting see [Client-Side Reporting](#).

Important

The `DEBUG` value is case-sensitive.

After you set the log mode to debug, we recommend that you verify that the debug session is active. To verify that the debug session is active, confirm that there's a `SESSION_INITIALIZED` event associated with the session ID in the CloudWatch logs. The session ID is included in the playback endpoint that MediaTailor provides. For more information, see [Verify that the debug log mode is active for your playback session](#).

Maximum active debug sessions

You can have a maximum of 10 active debug log sessions. When your player sends its session initialization or playback request to MediaTailor, MediaTailor checks to see if the limit has been reached. If it has, MediaTailor checks to see if there are any stale sessions. A session is stale if it hasn't been accessed within a certain period of time. For live streams this time period is 10 minutes, for VOD streams it's 30 minutes.

If the maximum active debug log sessions limit has been reached, debug logs aren't written to CloudWatch Logs for your session. If you don't see debug logs in CloudWatch Logs for your session, you could have reached this limit. To confirm if the limit has been reached, see [Verify that the debug log mode is active for your playback session](#).

Debug log fields

The following table lists the debug log fields that MediaTailor writes to CloudWatch.

Field	Description
<code>awsAccountId</code>	Your AWS account ID.
<code>customerId</code>	Your MediaTailor customer ID.
<code>eventTimestamp</code>	The ISO 8601 timestamp associated with the debug log event.
<code>eventType</code>	The type of debug log event. Values: • <code>ORIGIN_INTERACTION</code> – Details about MediaTailor interactions with the origin server. For example, the origin manifest response, manifest type, and origin URL. • <code>GENERATED_MANIFEST</code> – Details about the playback session response from MediaTailor. For example, the manifest that MediaTailor generates. • <code>SESSION_INITIALIZED</code> – Session initialization details, such as the session ID.
<code>originRequestUrl</code>	The URL of your origin server that is retrieved for this request.
<code>mediaTailorPath</code>	The MediaTailor endpoint that was called, including any parameters passed to MediaTailor in the initial manifest request.
<code>requestId</code>	The ID of a specific HTTP request to MediaTailor.
<code>responseBody</code>	The manifest in the response body from MediaTailor. This is either the raw origin manifest or the manifest generated by MediaTailor.
<code>sessionId</code>	The playback session ID.
<code>sessionType</code>	The type of playback session. Values: HLS, DASH

Read the debug logs

MediaTailor writes the debug logs to Amazon CloudWatch Logs. Typical CloudWatch Logs charges apply. Use CloudWatch Insights to read the debug logs. For information on how to use CloudWatch Logs Insights, see [Analyzing Log Data with CloudWatch Logs Insights](#) in the *AWS CloudWatch Logs User Guide*.

Note

The debug logs can take a few minutes to appear in CloudWatch. If you don't see the logs, wait a few minutes and try again. If you still don't see the logs, it could be that you've reached the maximum number of active debug log sessions. To verify if this is the case, run a CloudWatch query to see if there was a debug session initialized for your playback session. For more information, see [Verify that the debug log mode is active for your playback session](#).

Examples

This section includes example queries that you can use to read MediaTailor debug log data.

Example 1: Verify that the debug log mode is active for your playback session

```
fields @timestamp, @message
| filter sessionId = "32002de2-837c-4e3e-9660-f3075e8dfd90"
| filter eventType = "SESSION_INITIALIZED" # client-side reporting
or mediaTailorPath like "/v1/master" # server-side reporting HLS
or mediaTailorPath like "/v1/dash" # server-side reporting DASH
```

Example 2: View the responses from your origin

```
fields @timestamp, responseBody, @message, mediaTailorPath
| filter eventType = "ORIGIN_MANIFEST" and sessionId = "32002de2-837c-4e3e-9660-f3075e8dfd90"
```

Example 3: View the manifest generated by MediaTailor for a given session

```
fields @timestamp, responseBody, @message
| filter mediaTailorPath like "/v1/master/" and eventType = "GENERATED_MANIFEST" and
sessionId = "32002de2-837c-4e3e-9660-f3075e8dfd90"
```

Example 4: View all events for a given requestId

Use this query to view the origin manifest and the manifest generated by MediaTailor.

```
fields @timestamp, responseBody, @message, mediaTailorPath
| filter requestId = "e5ba82a5-f8ac-4efb-88a0-55bed21c45b4"
```

Monitoring AWS Elemental MediaTailor with Amazon CloudWatch metrics

You can monitor AWS Elemental MediaTailor metrics using CloudWatch. CloudWatch collects raw data about the performance of the service and processes that data into readable, near real-time metrics. These statistics are kept for 15 months, so that you can access historical information and gain a better perspective on how your web application or service is performing. You can also set alarms that watch for certain thresholds, and send notifications or take actions when those thresholds are met. For more information, see the [Amazon CloudWatch User Guide](#).

Metrics can be useful when you investigate stale manifests. For more information, see [Using metrics to diagnose stale manifests](#).

Metrics are grouped first by the service namespace, and then by the various dimension combinations within each namespace.

To view metrics using the CloudWatch console

1. Open the CloudWatch console at <https://console.aws.amazon.com/cloudwatch/>.
2. In the navigation pane, choose **Metrics**.
3. Under **All metrics**, choose the **MediaTailor** namespace.
4. Select the metric dimension to view the metrics (for example, **originID**).
5. Specify the time period that you want to view.

To view metrics using the AWS Command Line Interface (AWS CLI)

- At a command prompt, use the following command:

```
aws cloudwatch list-metrics --namespace "AWS/MediaTailor"
```

AWS Elemental MediaTailor CloudWatch metrics

The AWS Elemental MediaTailor namespace includes the following metrics. These metrics are published by default to your account.

Channel Assembly (CA) metrics

In the following table, all metrics are available by channel or by channel output.

Metric	Description
4xxErrorCount	The number of 4xx errors.
5xxErrorCount	The number of 5xx errors.
RequestCount	The total number of requests. The transaction count depends largely on how often players request updated manifests, and the number of players. Each player request counts as a transaction.
TotalTime	The amount of time that the application server took to process the request, including the time used to receive bytes from and write bytes to the client and network.

Server-side Ad-insertion (SSAI) metrics

The following table lists server-side ad-insertion metrics.

Metric	Description
AdDecisionServer.Ads	The count of ads included in ad decision server (ADS) responses within the CloudWatch time period that you specified. Dimensions: ConfigurationName

Metric	Description
	Account-level (no dimensions)
<code>AdDecisionServer.Duration</code>	The total duration, in milliseconds, of all ads that MediaTailor received from the ADS within the CloudWatch time period that you specified. This duration can be greater than the <code>Avail.Duration</code> that you specified. Dimensions: <code>ConfigurationName</code>
<code>AdDecisionServer.Errors</code>	The number of non-HTTP 200 status code responses, empty responses, and timed-out responses that MediaTailor received from the ADS within the CloudWatch time period that you specified. Dimensions: <code>ConfigurationName</code>
<code>AdDecisionServer.FillRate</code>	The simple average of the rates at which the responses from the ADS filled the corresponding individual ad avails for the time period that you specified. To get the weighted average, calculate the <code>AdDecisionServer.Duration</code> as a percentage of the <code>Avail.Duration</code>. For more information about simple and weighted averages, see Simple and weighted averages. Dimensions: <code>ConfigurationName</code>
<code>AdDecisionServer.Latency</code>	The response time in milliseconds for requests made by MediaTailor to the ADS. Dimensions: <code>ConfigurationName</code>

Metric	Description
AdDecisionServer.Timeouts	The number of timed-out requests to the ADS in the CloudWatch time period that you specified. Dimensions: ConfigurationName
AdNotReady	The number of times that the ADS pointed at an ad that wasn't yet transcoded by the internal transcoder service in the time period that you specified. A high value for this metric might contribute to a low overall Avail.FillRate . Dimensions: ConfigurationName
AdsBilled	The number of ads for which MediaTailor bills customers based on insertion. Dimensions: • ConfigurationName • Account-level (no dimensions)
Avail.Duration	The planned total number of milliseconds of ad avails within the CloudWatch time period. The planned total is based on the ad avail durations in the origin manifest. Dimensions: • ConfigurationName • Account-level (no dimensions)

Metric	Description
<code>Avail.FilledDuration</code>	The planned number of milliseconds of ad avail time that MediaTailor will fill with ads within the CloudWatch time period. Dimensions: • <code>ConfigurationName</code> • Account-level (no dimensions)
<code>Avail.FillRate</code>	The planned simple average of the rates at which MediaTailor will fill individual ad avails within the CloudWatch time period. To get the weighted average, calculate the <code>Avail.FilledDuration</code> as a percentage of the <code>Avail.Duration</code>. For more information about simple and weighted averages, see Simple and weighted averages. The maximum <code>Avail.FillRate</code> that MediaTailor can attain is bounded by the <code>AdDecisionServer.FillRate</code>. If the <code>Avail.FillRate</code> is low, compare it to the <code>AdDecisionServer.FillRate</code>. If the <code>AdDecisionServer.FillRate</code> is low, your ADS might not be returning enough ads for the avail durations. Dimensions: <code>ConfigurationName</code>

Metric	Description
Avail.Impression	The number of ads for which MediaTailor fired impression beacons during server-side reporting. MediaTailor emits one count for each ad that has at least one impression URI, regardless of whether those beacons were ultimately delivered. Use this metric as the denominator to calculate the video completion rate (VCR) (see Avail.Complete). Dimensions: • ConfigurationName • Account-level (no dimensions)
Avail.Complete	The number of ads for which MediaTailor fired complete beacons during server-side reporting . MediaTailor emits one count for each ad that has at least one complete tracking URI, regardless of whether those beacons were ultimately delivered. To calculate the video completion rate (VCR), divide the sum of Avail.Complete by the sum of Avail.Impression . Dimensions: • ConfigurationName • Account-level (no dimensions)

Metric	Description
Avail.ObservedDuration	The observed total number of milliseconds of ad avails that occurred within the CloudWatch time period. Avail.ObservedDuration is emitted at the end of the ad avail, and is based on the duration of the segments reported in the manifest during the ad avail. Dimensions: ConfigurationName
Avail.ObservedFilledDuration	The observed number of milliseconds of ad avail time that MediaTailor filled with ads within the CloudWatch time period. Dimensions: ConfigurationName
Avail.ObservedFillRate	The observed simple average of the rates at which MediaTailor filled individual ad avails within the CloudWatch time period. Emitted only for HLS manifests, at the first CUE-IN tag. If there is no CUE-IN tag, MediaTailor doesn't emit this metric. Dimensions: ConfigurationName
Avail.ObservedSlateDuration	The observed total number of milliseconds of slate that was inserted within the CloudWatch period. Dimensions: ConfigurationName

Metric	Description
GetManifest.Age	The total age of the manifest in milliseconds. Measured from when the origin creates the manifest, to when MediaTailor sends the personalized manifest. For more information about metrics for measuring manifest age, see Using metrics to diagnose stale manifests. Dimensions: ConfigurationName
GetManifest.Errors	The number of errors received while MediaTailor was generating manifests in the CloudWatch time period that you specified. Dimensions: ConfigurationName
GetManifest.Latency	The MediaTailor response time in milliseconds for the request to generate manifests. For more information about metrics for measuring manifest age, see Using metrics to diagnose stale manifests. Dimensions: ConfigurationName
GetManifest.MediaTailorAge	The amount of time that the manifest has been stored in MediaTailor in milliseconds. Measured from when MediaTailor receives an origin response, to when MediaTailor sends the personalized manifest. For more information about metrics for measuring manifest age, see Using metrics to diagnose stale manifests. Dimensions: ConfigurationName

Metric	Description
Origin.Age	The amount of time that the origin has the manifest in milliseconds. Measured from when the origin creates the manifest, to when MediaTailor sends the origin request. All <code>origin.*</code> metrics are emitted for requests that are fulfilled directly from the origin. They are not emitted for cached origin responses. For more information about metrics for measuring manifest age, see Using metrics to diagnose stale manifests. Dimensions: ConfigurationName
Origin.Errors	The number of non-HTTP 200 status code responses and timed-out responses that MediaTailor received from the origin server in the CloudWatch time period that you specified . All <code>origin.*</code> metrics are emitted for requests that are fulfilled directly from the origin. They are not emitted for cached origin responses. Dimensions: ConfigurationName
Origin.ManifestFileSizeBytes	The file size of the origin manifest in bytes for both HLS and DASH. Typically this metric is used in conjunction with <code>Origin.ManifestFileSizeTooLarge</code> . All <code>origin.*</code> metrics are emitted for requests that are fulfilled directly from the origin. They are not emitted for cached origin responses. Dimensions: ConfigurationName

Metric	Description
Origin.ManifestFileSizeTooLarge	The number of responses from the origin that have a manifest size larger than the configured amount. Typically this metric is used in conjunction with <code>Origin.ManifestFileSizeBytes</code>. All <code>origin.*</code> metrics are emitted for requests that are fulfilled directly from the origin. They are not emitted for cached origin responses. Dimensions: ConfigurationName
Origin.Timeouts	The number of timed-out requests to the origin server in the CloudWatch time period that you specified. All <code>origin.*</code> metrics are emitted for requests that are fulfilled directly from the origin. They are not emitted for cached origin responses. Dimensions: ConfigurationName
Requests	The number of concurrent transactions per second across all request types. The transaction count depends mainly on the number of players and how often the players request updated manifests. Each player request counts as a transaction. Dimensions: - ConfigurationName - Account-level (no dimensions)

Metric	Description
SkippedReason.DurationExceeded	The number of ads that were not inserted into an avail because the ADS returned a duration of ads that was greater than the specified avail duration. A high value for this metric might contribute to a discrepancy between the <code>AdsBilled</code> and <code>AdDecisionServer.Ads</code> metrics. For more information about ad skipped reasons, see Ad skipping troubleshooting. Dimensions: • <code>ConfigurationName</code> • Account-level (no dimensions)
SkippedReason.EarlyCueIn	The number of ads skipped due to an early CUE-IN. Dimensions: • <code>ConfigurationName</code> • Account-level (no dimensions)
SkippedReason.ImportError	The number of ads skipped due to an error in the import job. Dimensions: • <code>ConfigurationName</code> • Account-level (no dimensions)

Metric	Description
SkippedReason.ImportInProgress	The number of ads skipped due to an existing active import job. Dimensions: • ConfigurationName • Account-level (no dimensions)
SkippedReason.InternalError	The number of ads skipped due to a MediaTailor or internal error. Dimensions: • ConfigurationName • Account-level (no dimensions)
SkippedReason.NewCreative	The number of ads that were not inserted into an avail because it was the first time the asset had been requested by a client. A high value for this metric might temporarily contribute to a low overall <code>Avail.FillRate</code> , until assets can be successfully transcoded. Dimensions: • ConfigurationName • Account-level (no dimensions)
SkippedReason.NoVariantMatch	The number of ads skipped due to there being no variant match between the ad and content. Dimensions: • ConfigurationName • Account-level (no dimensions)

Metric	Description
SkippedReason.PersonalizationThresholdExceeded	The duration of ads exceeding the Personalization Threshold setting in this configuration. Dimensions: • ConfigurationName • Account-level (no dimensions)
SkippedReason.ProfileNotFound	The number of ads skipped due to the transcoding profile not being found. Dimensions: • ConfigurationName • Account-level (no dimensions)
SkippedReason.TranscodeError	The number of ads skipped due to a transcode error. Dimensions: • ConfigurationName • Account-level (no dimensions)
SkippedReason.TranscodeInProgress	The count of the number of ads that were not inserted into an avail because the ad had not yet been transcoded. A high value for this metric might temporarily contribute to a low overall <code>Avail.FillRate</code>, until the assets can be successfully transcoded. Dimensions: • ConfigurationName • Account-level (no dimensions)

Note

For HLS Interstitials sessions, some metrics behave differently due to the late-binding nature of ad decisioning:

- `Avail.ObservedFilledDuration` matches `Avail.FilledDuration` since MediaTailor cannot observe actual client-side playback behavior.
- `Avail.ObservedSlateDuration` reports planned slate duration from Asset List responses rather than observed playback.
- Metrics prefixed with "Observed" provide estimated values for HLS Interstitials sessions.

Functions metrics

The following metrics are published when you use [Functions](#) with your playback configurations. These metrics are always emitted when a function is configured on a lifecycle hook. No opt-in or log configuration is required.

Hook-level metrics — one data point per lifecycle hook execution:

Metric	Description
<code>PreSessionInitHook.Invocations</code>	The number of times the pre-session initialization hook was invoked. Dimensions: ConfigurationName
<code>PreSessionInitHook.Errors</code>	The number of pre-session initialization hook executions that resulted in an error. Dimensions: ConfigurationName
<code>PreSessionInitHook.Latency</code>	The execution time in milliseconds for the pre-session initialization hook. Dimensions: ConfigurationName
<code>PreAdsRequestHook.Invocations</code>	The number of times the pre-ads request hook was invoked.

Metric	Description
	Dimensions: ConfigurationName
PreAdsRequestHook.Errors	The number of pre-ads request hook executions that resulted in an error. Dimensions: ConfigurationName
PreAdsRequestHook.Latency	The execution time in milliseconds for the pre-ads request hook. Dimensions: ConfigurationName

Function-level metrics — one data point per individual function execution. These metrics include additional dimensions (FunctionId, FunctionType, HookType) so you can identify which specific function is slow or failing:

Metric	Description
Function.Invocations	The number of times an individual function was executed. Dimensions: - ConfigurationName - ConfigurationName , FunctionId , FunctionType , HookType
Function.Errors	The number of individual function executions that resulted in an error. Dimensions: - ConfigurationName - ConfigurationName , FunctionId , FunctionType , HookType

Metric	Description
Function.Latency	The execution time in milliseconds for an individual function. Dimensions: • ConfigurationName • ConfigurationName , FunctionId , FunctionType , HookType

Simple and weighted averages

You can retrieve the simple average and the weighted average for the responses from the ADS to ad requests from MediaTailor and for how MediaTailor fills ad avails:

- The *simple averages* are provided in the `AdDecisionServer.FillRate` and the `Avail.FillRate`. These are the averages of the fill rate percentages for the individual avails for the time period. The simple averages don't take into account any differences between the durations of the individual avails.
- The *weighted averages* are the fill rate percentages for the sum of all avail durations. These are calculated as $(\text{AdDecisionServer.Duration} \times 100) / \text{Avail.Duration}$ and $(\text{Avail.FilledDuration} \times 100) / \text{Avail.Duration}$. These averages reflect the differences in duration of each ad avail, giving more weight to those with longer duration.

For a time period that contains just a single ad avail, the simple average provided by the `AdDecisionServer.FillRate` is equal to the weighted average provided by $(\text{AdDecisionServer.Duration} \times 100) / \text{Avail.Duration}$. The simple average provided by the `Avail.FillRate` is equal to the weighted average provided by $(\text{Avail.FilledDuration} \times 100) / \text{Avail.Duration}$.

Example

Assume the time period that you specified has the following two ad avails:

- The first ad avail has 90 seconds duration:
 - The ADS response for the avail provides 45 seconds of ads (50% filled).

- MediaTailor fills 45 seconds worth of the ad time available (50% filled).
- The second ad avail has 120 seconds duration:
 - The ADS response for the avail provides 120 seconds of ads (100% filled).
 - MediaTailor fills 90 seconds worth of the ad time available (75% filled).

The metrics are as follows:

- `Avail.Duration` is 210, the sum of the two ad avail durations: $90 + 120$.
- `AdDecisionServer.Duration` is 165, the sum of the two response durations: $45 + 120$.
- `Avail.FilledDuration` is 135, the sum of the two filled durations: $45 + 90$.
- `AdDecisionServer.FillRate` is 75%, the average of the percentages filled for each avail: $(50\% + 100\%) / 2$. This is the simple average.
- The weighted average for the ADS fill rates is 78.57%, which is `AdDecisionServer.Duration` as a percentage of the `Avail.Duration`: $(165 \times 100) / 210$. This calculation accounts for the differences in the durations.
- `Avail.FillRate` is 62.5%, the average of the filled percentages for each avail: $(50\% + 75\%) / 2$. This is the simple average.
- The weighted average for the MediaTailor avail fill rates is 64.29%, which is the `Avail.FilledDuration` as a percentage of the `Avail.Duration`: $(135 \times 100) / 210$. This calculation accounts for the differences in the durations.

The highest `Avail.FillRate` that MediaTailor can attain for any ad avail is 100%. The ADS might return more ad time than is available in the avail, but MediaTailor can only fill the time available.

AWS Elemental MediaTailor CloudWatch dimensions

You can filter the AWS Elemental MediaTailor data using the following dimensions.

Dimension	Description
<code>ConfigurationName</code>	Indicates the configuration that the metric belongs to. Available on all metrics.
<code>FunctionId</code>	The identifier of the function. Available on <code>Function.*</code> metrics only.

Dimension	Description
FunctionType	The type of function: CUSTOM_OUTPUT , HTTP_REQUEST , SEQUENTIAL_EXECUTOR , or CONCURRENT_EXECUTOR . Available on Function.* metrics only.
HookType	The lifecycle hook that triggered the function: PRE_SESSION_INITIALIZATION or PRE_ADS_REQUEST . Available on Function.* metrics only.

Using metrics to diagnose stale manifests from AWS Elemental MediaTailor

A stale manifest is one that hasn't been recently updated. Different ad insertion workflows could have varying tolerance to how long must pass before a manifest is considered stale, based on a variety of factors (such as requirements for downstream systems). You can use Amazon CloudWatch metrics to identify manifests that exceed the staleness tolerance for your workflow, and help identify what could be causing the delays in manifest updates.

The following metrics help identify stale manifests and their causes.

For information about all metrics that MediaTailor emits, see [AWS Elemental MediaTailor CloudWatch metrics](#).

Metric	Definition	Use
GetManifest.Age	Measures the total age of the manifest, including both GetManifest.MediaTailorAge and Origin.Age for this configuration.	You can use this metric to identify manifests that are past your update threshold and are stale. Set alarms on this metric so that you're alerted when stale manifests are being served. For information about alarms, see Alarming on metrics in the <i>Amazon CloudWatch</i>

Metric	Definition	Use
		<i>h User Guide.</i> When you receive an alarm, use <code>Origin.Age</code> and <code>GetManifest.MediaTailorAge</code> to identify if MediaTailor or the origin is causing the staleness.
<code>Origin.Age</code>	Measures how long the origin has the manifest before sending it to MediaTailor for this configuration. This metric is not emitted when the response comes from a content delivery network (CDN). The response must come from the origin for <code>Origin.Age</code> to be emitted.	When you identify stale manifests with <code>GetManifest.Age</code> , you can analyze the <code>Origin.Age</code> metric and the <code>GetManifest.MediaTailorAge</code> metric to determine which is contributing to manifest staleness. If you find that <code>Origin.Age</code> is longer than your typical processing times at the origin, it's likely that the upstream system is causing the issue and you should focus diagnostics there.

Metric	Definition	Use
GetManifest.MediaTailorAge	Measures how long MediaTailor has stored this manifest for this configuration.	When you identify stale manifests with <code>GetManifest.Age</code> , you can analyze the <code>GetManifest.MediaTailorAge</code> metric and the <code>Origin.Age</code> metric to determine which is contributing to manifest staleness. If the <code>GetManifest.MediaTailorAge</code> is longer than your typical manifest personalization times in MediaTailor, it's likely that MediaTailor is causing the issue and you should focus diagnostics there. <code>GetManifest.Latency</code> can further identify how long it takes for MediaTailor to create a personalized manifest.

Metric	Definition	Use
GetManifest.Latency	Measures the amount of time it takes for MediaTailor to process the request and create a personalized manifest for this configuration.	When you compare <code>Origin.Age</code> and <code>GetManifest.MediaTailorAge</code> and determine that MediaTailor is the cause of delayed manifest delivery, you can analyze the <code>GetManifest.Latency</code> metric to determine if the manifest personalization process is contributing to manifest staleness. <code>GetManifest.MediaTailorAge</code> measures the total time that the manifest is stored in MediaTailor. <code>GetManifest.Latency</code> measures how much of that storage time is MediaTailor personalizing the manifest in response to a request.

Recording AWS Elemental MediaTailor API calls

AWS Elemental MediaTailor is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in MediaTailor. CloudTrail captures all API calls for MediaTailor as events. The calls captured include calls from the MediaTailor console and code calls to the MediaTailor API operations. If you create a trail, you can enable continuous delivery of CloudTrail events to an Amazon S3 bucket, including events for MediaTailor. If you don't configure a trail, you can still view the most recent events in the CloudTrail console in **Event history**. Using the information collected by CloudTrail, you can determine the request that was made to MediaTailor, the IP address from which the request was made, who made the request, when it was made, and additional details.

To learn more about CloudTrail, see the [AWS CloudTrail User Guide](#).

AWS Elemental MediaTailor information in CloudTrail

CloudTrail is enabled on your AWS account when you create the account. When activity occurs in AWS Elemental MediaTailor, that activity is recorded in a CloudTrail event along with other AWS service events in **Event history**. You can view, search, and download recent events in your AWS account. For more information, see [Viewing events with CloudTrail event history](#).

For an ongoing record of events in your AWS account, including events for AWS Elemental MediaTailor, create a trail. A *trail* enables CloudTrail to deliver log files to an Amazon S3 bucket. By default, when you create a trail in the console, the trail applies to all AWS Regions. The trail logs events from all Regions in the AWS partition and delivers the log files to the Amazon S3 bucket that you specify. Additionally, you can configure other AWS services to further analyze and act upon the event data collected in CloudTrail logs. For more information, see the following:

- [Creating a trail for your AWS account](#)
- [AWS service integrations with CloudTrail logs](#)
- [Configuring Amazon SNS notifications for CloudTrail](#)
- [Receiving CloudTrail log files from multiple regions](#) and [Receiving CloudTrail log files from multiple accounts](#)

All AWS Elemental MediaTailor actions are logged by CloudTrail and are documented in the [AWS Elemental MediaTailor API reference](#). For example, calls to the `PutPlaybackConfiguration` and `ListPlaybackConfigurations` operations generate entries in the CloudTrail log files.

Every event or log entry contains information about who generated the request. The identity information helps you determine the following:

- Whether the request was made with the root user or AWS Identity and Access Management (IAM) credentials
- Whether the request was made with temporary security credentials for a role or federated user
- Whether the request was made by another AWS service

For more information, see [CloudTrail userIdentity element](#).

Understanding AWS Elemental MediaTailor log file entries

A trail is a configuration that enables delivery of events as log files to an Amazon S3 bucket that you specify. CloudTrail log files contain one or more log entries. An event represents a single request from any source and includes information about the requested action, the date and time of the action, request parameters, and so on. CloudTrail log files aren't an ordered stack trace of the public API calls, so they don't appear in any specific order.

The following example shows a CloudTrail log entry that demonstrates the `PutPlaybackConfiguration` action:

```
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "IAMUser",
    "principalId": "AIDAEXAMPLE",
    "arn": "arn:aws:iam::111122223333:user/testuser",
    "accountId": "111122223333",
    "accessKeyId": "AIDAEXAMPLE",
    "userName": "testuser"
  },
  "eventTime": "2018-12-28T22:53:46Z",
  "eventSource": "mediatailor.amazonaws.com",
  "eventName": "PutPlaybackConfiguration",
  "awsRegion": "us-west-2",
  "sourceIPAddress": "1.2.3.4",
  "userAgent": "PostmanRuntime/7.4.0",
  "requestParameters": {
    "VideoContentSourceUrl": "http://examplevideo.com",
    "Name": "examplename",
    "AdDecisionServerUrl": "http://exampleads.com"
  },
  "responseElements": {
    "SessionInitializationEndpointPrefix": "https://777788889999.mediatailor.us-east-1.amazonaws.com/v1/session/AKIAIOSFODNN7EXAMPLE/examplename/",
    "DashConfiguration": {
      "ManifestEndpointPrefix": "https://777788889999.mediatailor.us-east-1.amazonaws.com/v1/dash/AKIAIOSFODNN7EXAMPLE/examplename/",
      "MpdLocation": "EMT_DEFAULT"
    },
    "AdDecisionServerUrl": "http://exampleads.com",
    "CdnConfiguration": {},
  }
}
```



```

    "PlaybackEndpointPrefix": "https://777788889999.mediatailor.us-east-1.amazonaws.com",
    "HlsConfiguration": {
        "ManifestEndpointPrefix": "https://777788889999.mediatailor.us-east-1.amazonaws.com/v1/master/AKIAIOSFODNN7EXAMPLE/exemplename/"
    },
    "VideoContentSourceUrl": "http://examplevideo.com",
    "Name": "exemplename"
},
"requestID": "1a2b3c4d-1234-5678-1234-1a2b3c4d5e6f",
"eventID": "987abc65-1a2b-3c4d-5d6e-987abc654def",
"readOnly": false,
"eventType": "AwsApiCall",
"recipientAccountId": "111122223333"
}

```

The following example shows a CloudTrail log entry that demonstrates the `GetPlaybackConfiguration` action:

```

{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "IAMUser",
    "principalId": "AIDAEXAMPLE",
    "arn": "arn:aws:iam::111122223333:user/testuser",
    "accountId": "111122223333",
    "accessKeyId": "AIDAEXAMPLE",
    "userName": "testuser"
  },
  "eventTime": "2018-12-28T22:52:37Z",
  "eventSource": "mediatailor.amazonaws.com",
  "eventName": "GetPlaybackConfiguration",
  "awsRegion": "us-west-2",
  "sourceIPAddress": "1.2.3.4",
  "userAgent": "PostmanRuntime/7.4.0",
  "requestParameters": {
    "Name": "exemplename"
  },
  "responseElements": null,
  "requestID": "0z1y2x3w-0123-4567-9876-6q7r8s9t0u1v",
  "eventID": "888ddd77-3322-eeew-uuii-abc123jkl343",
  "readOnly": true,
  "eventType": "AwsApiCall",

```

```
"recipientAccountId": "111122223333"  
}
```

Receiving AWS Elemental MediaTailor channel assembly alerts

MediaTailor creates alerts for issues or potential issues that occur with your channel assembly resources. The alert describes the issue, when the issue occurred, and the affected resources.

You can view the alerts in the AWS Management Console, the AWS Command Line Interface (AWS CLI), AWS SDKs, or programmatically using the MediaTailor [ListAlerts](#) API.

Important

Alerts are only available for channel assembly resources created on or after July 14th, 2021.

Channel Assembly Alerts

Alert Type	Alert Code	Alert Message	Notes
VOD Source	NOT_PROCESSED	MediaTailor hasn't processed the package configuration <i>configurationPath</i> .	
	UNREACHABLE	We can't reach the URL <i>url</i> .	
	UNAUTHORIZED	<i>url</i> didn't authorize the request.	
	TIMEOUT	The connection to <i>url</i> timed out.	
	UNPARSABLE_MANIFEST	MediaTailor encountered an issue while parsing the manifest from <i>url</i> .	

Alert Type	Alert Code	Alert Message	Notes
	VARIANT_DURATION_MISMATCH	MediaTailor encountered variants with mismatched total durations while parsing the manifest from <i>url</i> . This might cause stalling during playback.	Your manifest has varying durations across variants/representations. This might result in missing or incorrect captions, and MediaTailor being unable to insert ads.
	SEGMENT_DURATION_TOO_LONG	MediaTailor encountered a segment with a duration greater than thirty seconds while parsing the manifest from <i>url</i> . This might cause stalling during playback, missing or incorrect captions, and the inability to insert advertisements.	Your manifest contains a segment that is greater than 30 seconds.
	TARGET_DURATION_MISMATCH	MediaTailor encountered a mismatch of EXT-X-TARGETDURATION values across HLS manifests while parsing the manifest from <i>url</i> . This might cause stalling during playback.	The target duration doesn't match across all manifests in the source..
Source Location	NOT_PROCESSED	MediaTailor hasn't processed the resource <i>resourceName</i> .	
Program	VOD_SOURCE_ALERT	The VOD source <i>vodSourceName</i> in this program has the following alert: <i>vodSourceAlertCode</i> : <i>vodSourceAlertMessage</i>	

Alert Type	Alert Code	Alert Message	Notes
	SOURCE_LOCATION_ALERT	The source location <i>sourceLocationName</i> contained in this program has the following alert: <i>sourceLocationAlertCode : sourceLocationAlertMessage</i>	
	CODEC_MISMATCH	MediaTailor encountered mismatched codec in <i>channelName</i> schedule. The mismatch is in <i>sourceGroupName</i> between <i>programName1</i> 's manifest <i>manifestUrl</i> and <i>programName2</i> 's manifest <i>manifestUrl</i> .	
	RESOLUTION_MISMATCH	MediaTailor encountered mismatched resolution in <i>channelName</i> schedule. The mismatch is in <i>sourceGroupName</i> between <i>programName1</i> 's manifest <i>manifestUrl</i> and <i>programName2</i> 's manifest <i>manifestUrl</i> .	

Alert Type	Alert Code	Alert Message	Notes
	BANDWIDTH_MISMATCH	MediaTailor encountered mismatched bandwidth in <i>channelName</i> schedule. The mismatch is in <i>sourceGroupName</i> between <i>programName1</i> 's manifest <i>manifestUrl</i> and <i>programName2</i> 's manifest <i>manifestUrl</i> .	
	FRAMERATE_MISMATCH	MediaTailor encountered mismatched framerate in <i>channelName</i> schedule. The mismatch is in <i>sourceGroupName</i> between <i>programName1</i> 's manifest <i>manifestUrl</i> and <i>programName2</i> 's manifest <i>manifestUrl</i> .	
	TARGET_DURATION_MISMATCH	MediaTailor encountered mismatched EXT-X-TARGETDURATION values across HLS manifests in <i>channelName</i> schedule. The mismatch is in <i>sourceGroupName</i> between <i>programName1</i> 's manifest <i>manifestUrl</i> and <i>programName2</i> 's manifest <i>manifestUrl</i> .	

Alert Type	Alert Code	Alert Message	Notes
	SEGMENT_DURATION_MISMATCH	MediaTailor encountered mismatched segment duration values across manifests in <i>channelName</i> schedule. The mismatch is in <i>sourceGroupName</i> between <i>programName1</i> 's manifest <i>manifestUrl</i> and <i>programName2</i> 's manifest <i>manifestUrl</i> .	
	NO_COMMON_SEGMENT_BOUNDARY_FOR_AD_SLATE	MediaTailor was not able to insert ad slate at offset <i>offsetMillis</i> for program <i>programName</i> . There is no common segment boundary at the ad slate's start time.	
	NOT_PROCESSED	MediaTailor hasn't processed the resource <i>resourceName</i> .	
	TOO_MANY_ALERTS	MediaTailor has found too many alerts and will not provide any more alerts for <i>programName</i> . Clear existing alerts to continue receiving alerts for <i>programName</i> .	
Channel	PROGRAM_ALERT	The program <i>programName</i> contained in this channel has the following alert: <i>programAlertCode</i> : <i>programAlertMessage</i>	

Troubleshooting manifest-related alerts

When you encounter manifest-related alerts such as UNPARSABLE_MANIFEST, VARIANT_DURATION_MISMATCH, or TARGET_DURATION_MISMATCH, check the following:

1. **Verify source requirements:** Ensure your content meets the requirements outlined in [the section called “Integrating a content source”](#).
2. **Check manifest logs:** For detailed manifest error analysis and event types, see [the section called “Manifest logs”](#).
3. **Check error codes:** For detailed error analysis, see [Troubleshooting](#).
4. **Review manifest format:** Verify that your manifest follows the correct format specifications for your streaming protocol (HLS or DASH).

Viewing alerts

You can view alerts for any MediaTailor channel assembly resource. When you view the alerts for channels and programs, MediaTailor includes all of the related resources contained within the channel or program. For example, when you view the alerts for a specific program, you also see alerts for the source location and VOD sources that the program contains.

To view alerts, perform the following procedure.

Console

To view alerts in the console

1. Open the MediaTailor console at <https://console.aws.amazon.com/mediatailor/>.
2. Choose the resource that you want to view alerts for.
3. Select the **Alerts** tab to view the alerts.

AWS Command Line Interface (AWS CLI)

To list alerts for a channel assembly resource, you need the resource's [Amazon Resource Name \(ARN\)](#). You can use the `describe-resource_type` command in the AWS Command Line Interface (AWS CLI) to get the resource's ARN. For example, run the [describe-channel](#) command to get a specific channel's ARN:

```
aws mediatailor describe-channel --channel-name MyChannelName
```

Then use the [aws mediatailor list-alerts](#) command to list the alerts associated with the resource:

```
aws mediatailor list-alerts --resource-arn arn:aws:mediatailor:region:aws-account-id:resource-type/resource-name
```

API

To list alerts for a channel assembly resource, you need the resource's [Amazon Resource Name \(ARN\)](#). You can use the `DescribeResource` operation in the MediaTailor API to get the resource's ARN. For example, use the [DescribeChannel](#) operation to get a specific channel's ARN.

Then use the [ListAlerts](#) API to list the alerts for the resource.

Handling alerts

When an alert occurs, view the alerts in the AWS Management Console, or use the AWS Command Line Interface (AWS CLI), AWS SDKs, or the MediaTailor Alerts API to determine the possible sources of the issue.

After you resolve the issue, MediaTailor clears the alert.

Tagging AWS Elemental MediaTailor resources

A *tag* is a metadata label that you assign or that AWS assigns to an AWS resource. Each tag consists of a *key* and a *value*. For tags that you assign, you define the key and value. For example, you might define the key as `stage` and the value for one resource as `test`.

Tags help you do the following:

- Identify and organize your AWS resources. Many AWS services support tagging, so you can assign the same tag to resources from different services to indicate that the resources are related. For example, you could assign the same tag to an AWS Elemental MediaPackage channel and endpoint that you assign to an AWS Elemental MediaTailor configuration.
- Track your AWS costs. You activate these tags on the AWS Billing and Cost Management dashboard. AWS uses the tags to categorize your costs and deliver a monthly cost allocation report to you. For more information, see [Use cost allocation tags](#) in the [AWS Billing User Guide](#).

- Control access to your AWS resources. For more information, see [Controlling access using tags](#) in the [IAM User Guide](#).

The following sections provide more information about tags for AWS Elemental MediaTailor.

Supported resources in AWS Elemental MediaTailor

The following resources in AWS Elemental MediaTailor supports tagging:

- Channels
- Configurations
- SourceLocations
- VodSources

Tag restrictions

The following basic restrictions apply to tags on AWS Elemental MediaTailor resources:

- Maximum number of tags that you can assign to a resource – 50
- Maximum key length – 128 Unicode characters
- Maximum value length – 256 Unicode characters
- Valid characters for key and value – a-z, A-Z, 0-9, space, and the following characters: _ . : / = + - and @
- Keys and values are case sensitive
- Don't use `aws :` as a prefix for keys; it's reserved for AWS use

Managing tags in AWS Elemental MediaTailor

You set tags as properties on a resource. You can add, edit, and delete tags through the AWS Elemental MediaTailor API or the AWS Command Line Interface (AWS CLI). For more information, see the [AWS Elemental MediaTailor API reference](#).

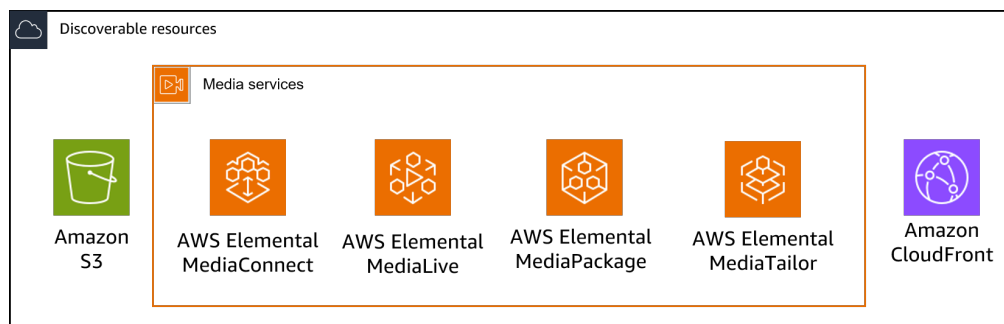
Monitoring AWS media services with workflow monitor

Workflow monitor is a tool for the discovery, visualization, and monitoring of AWS media workflows. Workflow monitor is available in the AWS console and API. You can use workflow

monitor to discover and create visual mappings of your workflow's resources, called *signal maps*. You can create and manage Amazon CloudWatch alarm and Amazon EventBridge rule templates to monitor the mapped resources. The monitoring templates you create are transformed into deployable AWS CloudFormation templates to allow repeatability. AWS recommended alarm templates provide predefined best-practice monitoring.

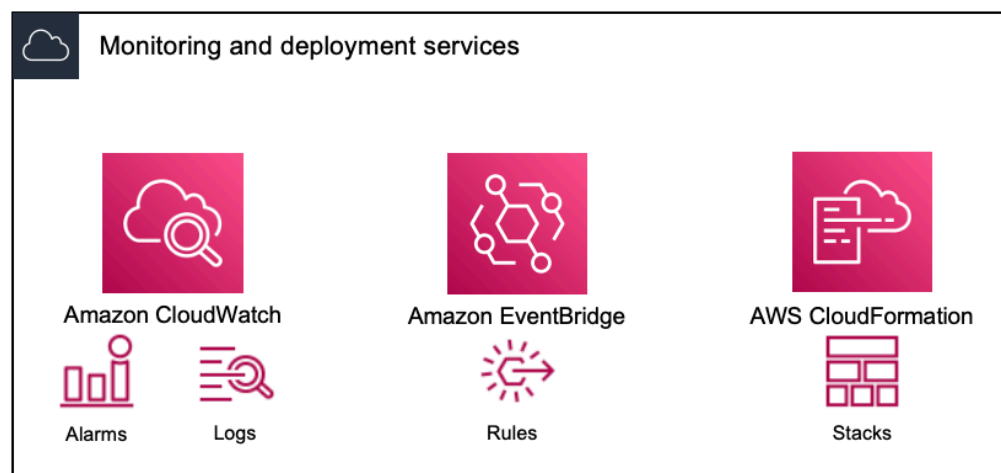
Discover

Utilize signal maps to automatically discover interconnected AWS resources associated with your media workflow. Discovery can begin at any supported service resource and creates an end-to-end mapping of the workflow. Signal maps can be used as stand-alone visualization tools or enhanced with monitoring templates.



Monitor

You can create custom CloudWatch alarm and EventBridge rule templates to monitor the health and status of your media workflows. Best practice alarm templates are available to import into your workflow monitor environment. You can use the best practice alarm templates as they are, or edit them to better fit your workflow. Any templates you create are transformed into CloudFormation templates for repeatable deployment.



Note

There is no direct cost for using workflow monitor. However, there are costs associated with the resources created and used to monitor your workflow.

When monitoring is deployed, Amazon CloudWatch and Amazon EventBridge resources are created. When using the AWS Management Console, prior to deploying monitoring to a signal map, you will be notified of how many resources will be created. For more information about pricing, see: [CloudWatch pricing](#) and [EventBridge pricing](#).

Workflow monitor uses AWS CloudFormation templates to deploy the CloudWatch and EventBridge resources. These templates are stored in a standard class Amazon Simple Storage Service bucket that is created on your behalf, by workflow monitor, during the deployment process and will incur object storage and recall charges. For more information about pricing, see: [Amazon S3 pricing](#).

Previews generated in the workflow monitor signal map for AWS Elemental MediaPackage channels are delivered from the MediaPackage Origin Endpoint and will incur Data Transfer Out charges. For pricing, see: [MediaPackage pricing](#).

Components of workflow monitor

Workflow monitor has four major components:

- CloudWatch alarm templates - Define the conditions you would like to monitor using CloudWatch. You can create your own alarm templates, or import predefined templates created by AWS. For more information, see: [CloudWatch alarm groups and templates for monitoring your AWS media workflow](#)
- EventBridge rule templates - Define how EventBridge sends notifications when an alarm is triggered. For more information, see: [EventBridge rule groups and templates for monitoring your AWS media workflow](#)
- Signal maps - Use an automated process to create AWS Elemental workflow maps using existing AWS resources. The signal maps can be used to discover resources in your workflow and deploy monitoring to those resources. For more information, see: [Workflow monitor signal maps](#)
- Overview - The overview page allows you to directly monitor the status of multiple signal maps from one location. Review metrics, logs, and alarms for your workflows. For more information, see: [Workflow monitor overview](#)

Supported services

Workflow monitor supports automatic discovery and signal mapping of resources associated with the following services:

- AWS Elemental MediaConnect
- AWS Elemental MediaLive
- AWS Elemental MediaPackage
- AWS Elemental MediaTailor
- Amazon S3
- Amazon CloudFront

Topics

- [Configuring workflow monitor to monitor AWS media services](#)
- [Using workflow monitor](#)

Configuring workflow monitor to monitor AWS media services

To setup workflow monitor for the first time; you create the alarm and event templates, and discover signal maps that are used to monitor your media workflows. The following guide contains the steps necessary to setup both Administrator and Operator level IAM roles, create workflow monitor resources, and deploy monitoring to your workflows.

Topics

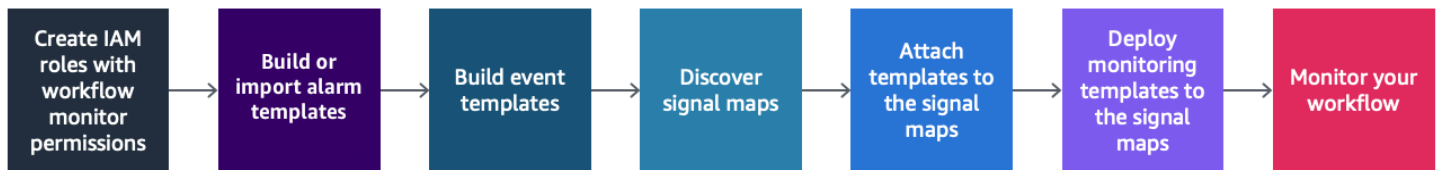
- [Getting started with workflow monitor](#)
- [Workflow monitor groups and templates](#)
- [Workflow monitor signal maps](#)
- [Workflow monitor quotas](#)

Getting started with workflow monitor

The following steps provide a basic overview of using workflow monitor for the first time.

1. Setup workflow monitor IAM permissions for administrator and operator level roles: [Workflow monitor IAM policies](#)

2. Build alarm templates or import predefined templates created by AWS: [CloudWatch alarms](#)
3. Build notification events that will be delivered by EventBridge: [EventBridge rules](#)
4. Discover signal maps using your existing AWS Elemental resources: [Signal maps](#)
5. Attach the alarm templates and notification rules to your signal map: [Attaching templates](#)
6. Deploy the templates to begin monitoring the signal map: [Deploying monitoring templates](#)
7. Monitor and review your workflow monitor resources using the overview section of the AWS console: [Overview](#)



Workflow monitor IAM policies

Workflow monitor interacts with multiple AWS services to create signal maps, build CloudWatch and EventBridge resources, and CloudFormation templates. Because workflow monitor interacts with a wide range of services, specific AWS Identity and Access Management (IAM) policies must be assigned for these services. The following examples indicate the necessary IAM policies for both administrator and operator IAM roles.

Administrator IAM policy

The following example policy is for an administrator-level workflow monitor IAM policy. This role allows for the creation and management of workflow monitor resources and the supported service resources that interact with workflow monitor.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "cloudwatch:List*",
        "cloudwatch:Describe*",

```

```

        "cloudwatch:Get*",
        "cloudwatch:PutAnomalyDetector",
        "cloudwatch:PutMetricData",
        "cloudwatch:PutMetricAlarm",
        "cloudwatch:PutCompositeAlarm",
        "cloudwatch:PutDashboard",
        "cloudwatch:DeleteAlarms",
        "cloudwatch:DeleteAnomalyDetector",
        "cloudwatch:DeleteDashboards",
        "cloudwatch:TagResource",
        "cloudwatch:UntagResource"
    ],
    "Resource": "*"
},
{
    "Effect": "Allow",
    "Action": [
        "cloudformation:List*",
        "cloudformation:Describe*",
        "cloudformation:CreateStack",
        "cloudformation:UpdateStack",
        "cloudformation:DeleteStack",
        "cloudformation:TagResource",
        "cloudformation:UntagResource"
    ],
    "Resource": "*"
},
{
    "Effect": "Allow",
    "Action": [
        "cloudfront:List*",
        "cloudfront:Get*"
    ],
    "Resource": "*"
},
{
    "Effect": "Allow",
    "Action": [
        "ec2:DescribeNetworkInterfaces"
    ],
    "Resource": "*"
},
{
    "Effect": "Allow",

```

```

    "Action": [
      "events:List*",
      "events:Describe*",
      "events:CreateEventBus",
      "events:PutRule",
      "events:PutTargets",
      "events:EnableRule",
      "events:DisableRule",
      "events>DeleteRule",
      "events:RemoveTargets",
      "events:TagResource",
      "events:UntagResource"
    ],
    "Resource": "*"
  },
  {
    "Effect": "Allow",
    "Action": [
      "logs:Describe*",
      "logs:Get*",
      "logs:TagLogGroup",
      "logs:TagResource",
      "logs:UntagLogGroup",
      "logs:UntagResource"
    ],
    "Resource": "*"
  },
  {
    "Effect": "Allow",
    "Action": [
      "mediaconnect:List*",
      "mediaconnect:Describe*"
    ],
    "Resource": "*"
  },
  {
    "Effect": "Allow",
    "Action": [
      "medialive:*"
    ],
    "Resource": "*"
  },
  {
    "Effect": "Allow",

```

```

    "Action": [
      "mediapackage:List*",
      "mediapackage:Describe*"
    ],
    "Resource": "*"
  },
  {
    "Effect": "Allow",
    "Action": [
      "mediapackagev2:List*",
      "mediapackagev2:Get*"
    ],
    "Resource": "*"
  },
  {
    "Effect": "Allow",
    "Action": [
      "mediapackage-vod:List*",
      "mediapackage-vod:Describe*"
    ],
    "Resource": "*"
  },
  {
    "Effect": "Allow",
    "Action": [
      "mediatailor:List*",
      "mediatailor:Describe*",
      "mediatailor:Get*"
    ],
    "Resource": "*"
  },
  {
    "Effect": "Allow",
    "Action": [
      "resource-groups:ListGroup",
      "resource-groups:GetGroup",
      "resource-groups:GetTags",
      "resource-groups:GetGroupQuery",
      "resource-groups:GetGroupConfiguration",
      "resource-groups:CreateGroup",
      "resource-groups:UngroupResources",
      "resource-groups:GroupResources",
      "resource-groups>DeleteGroup",
      "resource-groups:UpdateGroupQuery",

```



```

        "resource-groups:UpdateGroup",
        "resource-groups:Tag",
        "resource-groups:Untag"
    ],
    "Resource": "*"
},
{
    "Effect": "Allow",
    "Action": [
        "s3:*"
    ],
    "Resource": "arn:aws:s3:::workflow-monitor-templates*"
},
{
    "Effect": "Allow",
    "Action": [
        "sns:TagResource",
        "sns:UntagResource"
    ],
    "Resource": "*"
},
{
    "Effect": "Allow",
    "Action": [
        "tag:Get*",
        "tag:Describe*",
        "tag:TagResources",
        "tag:UntagResources"
    ],
    "Resource": "*"
}
]
}

```

Operator IAM policy

The following example policy is for an operator-level workflow monitor IAM policy. This role allows for limited and read-only access to the workflow monitor resources and the supported service resources that interact with workflow monitor.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "cloudwatch:List*",
        "cloudwatch:Describe*",
        "cloudwatch:Get*"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "cloudformation:List*",
        "cloudformation:Describe*"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "cloudfront:List*",
        "cloudfront:Get*"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "ec2:DescribeNetworkInterfaces"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "events:List*",
        "events:Describe*"
      ],
    },
```

```

    "Resource": "*"
  },
  {
    "Effect": "Allow",
    "Action": [
      "logs:Describe*",
      "logs:Get*"
    ],
    "Resource": "*"
  },
  {
    "Effect": "Allow",
    "Action": [
      "mediaconnect:List*",
      "mediaconnect:Describe*"
    ],
    "Resource": "*"
  },
  {
    "Effect": "Allow",
    "Action": [
      "medialive:List*",
      "medialive:Get*",
      "medialive:Describe*"
    ],
    "Resource": "*"
  },
  {
    "Effect": "Allow",
    "Action": [
      "mediapackage:List*",
      "mediapackage:Describe*"
    ],
    "Resource": "*"
  },
  {
    "Effect": "Allow",
    "Action": [
      "mediapackagev2:List*",
      "mediapackagev2:Get*"
    ],
    "Resource": "*"
  },
  {

```

```

    "Effect": "Allow",
    "Action": [
        "mediapackage-vod:List*",
        "mediapackage-vod:Describe*"
    ],
    "Resource": "*"
},
{
    "Effect": "Allow",
    "Action": [
        "mediatailor:List*",
        "mediatailor:Describe*",
        "mediatailor:Get*"
    ],
    "Resource": "*"
},
{
    "Effect": "Allow",
    "Action": [
        "s3:Get*",
        "s3:List*"
    ],
    "Resource": "arn:aws:s3:::workflow-monitor-templates*"
},
{
    "Effect": "Allow",
    "Action": [
        "tag:Get*",
        "tag:Describe*"
    ],
    "Resource": "*"
}
]
}

```

Workflow monitor groups and templates

Before you can deploy workflow monitoring to a signal map, you must create the groups and templates for CloudWatch alarms and EventBridge notifications. The CloudWatch templates define what scenarios and thresholds will be used to trigger the alarms. The EventBridge templates will determine how these alarms are reported to you.

If you only want mappings of your connected resources and do not want to use the monitoring template capabilities of workflow monitor, signal maps can be used without CloudWatch and EventBridge templates. For more information about using signal maps, see: [Signal maps](#)

Topics

- [CloudWatch alarm groups and templates for monitoring your AWS media workflow](#)
- [EventBridge rule groups and templates for monitoring your AWS media workflow](#)

CloudWatch alarm groups and templates for monitoring your AWS media workflow

Workflow monitor alarms allow you to use existing CloudWatch metrics as the foundation of alarms for your signal maps. You can create an alarm template group to sort and classify the types of alarming that is important to your workflow. Within each alarm template group, you create alarm templates with specific CloudWatch metrics and parameters that you want to monitor. You can create your own alarm templates or import recommended alarm templates created by AWS. After creating an alarm template group and alarm templates within that group, you can attach one or more of these alarm template groups to a signal map.

You must create an alarm template group first. After you have created an alarm template group, you can create your own templates or use recommended templates created by AWS. If you want to create your own alarm templates, continue on this page. For more information about importing recommended templates, see: [Recommended templates](#)

This section covers the creation of CloudWatch alarms using workflow monitor. For more information about how the CloudWatch service handles alarms and details of the alarm components, see: [Using CloudWatch alarms](#) in the *Amazon CloudWatch User Guide*

Creating alarm template groups

Alarm template groups allow you to sort and classify the types of alarms that are important to your workflow.

To create an alarm template group

1. From the workflow monitor console's navigation pane, select **CloudWatch alarm templates**.
2. Select **Create alarm template group**.
3. Give the alarm template group a unique **Group name** and optional **Description**.
4. Select **Create**, You will be taken to the newly created alarm template group's details page.

Creating alarm templates

You can create alarm templates with the CloudWatch metrics and parameters that you want to monitor.

To create an alarm template

1. From the alarm template group's details page, select **Create alarm template**.
2. Give the alarm template a unique **Template name** and optional **Description**.
3. In the **Choose metric** section:
 1. Select a **Target Resource Type**. The target resource type is a resource for the respective service, such as a channel for MediaLive and MediaPackage or a flow for MediaConnect.
 2. Select a **Metric Name**. This is the CloudWatch metric that acts as the foundation for the alarm. The list of metrics will change depending on the selected **Target Resource Type**.
4. In the **Alarm settings** section:

Note

For more information about how the CloudWatch service handles alarms and details of the alarm components, see: [Using CloudWatch alarms](#) in the *Amazon CloudWatch User Guide*

1. Select the **Statistic**. This is a value such as a **Sum** or an **Average** that will be used to monitor the metric.
2. Select the **Comparison Operator**. This field references the **Threshold** that you set in the next step.
3. Set a **Threshold**. This is a numeric value that the **Comparison Operator** uses to determine greater than, less than, or equal to status.
4. Set a **Period**. This is a time value, in seconds. The **Period** is the length of time that the **Statistic**, **Comparison Operator**, and **Threshold** interact to determine if the alarm gets triggered.
5. Set the **Datapoints**. This value determines how many datapoints are needed to trigger the alarm.

6. Select how to **Treat Missing Data**. This selection determines how this alarm reacts to missing data.
5. Select **Create** to complete the process.

An example of a completed alarm template could have the following parameters: A MediaConnect flow **Target Resource Type** is monitored for the Disconnections **Metric Name**. The **Statistic** value is set to Sum with a **Comparison Operator** of "greater than or equal to" and a **Threshold** of 10. The **Period** is set to 60 seconds, and only requires 1 out of 1 **Datapoints**. **Treat Missing Data** is set to "ignore."

The result of these settings is: workflow monitor will monitor for disconnections on the flow. If 10 or more disconnections occur within 60 seconds, the alarm will be triggered. 10 or more disconnections in 60 seconds only needs to happen one time for the alarm to be triggered.

Recommended alarm templates for monitoring your AWS media workflow

Workflow monitor's recommended templates are a curated selection of AWS Elemental service metrics with predefined alarm settings appropriate for the metric. If you do not want to create customized alarm templates, recommended templates provide you with best-practice monitoring templates that are created by AWS.

Workflow monitor contains recommended template groups for each supported service. These groups are designed to apply best-practice monitoring to specific types of workflows. Each template group contains a curated selection of alarms configured from service-specific metrics. For example, a recommended template group for a MediaLive multiplex workflow will have a different set of preconfigured metrics than a MediaConnect CDI workflow.

To use recommended alarm templates

1. Follow the steps to [create an alarm template group](#), or select an existing one.
2. In the **Alarm templates** section, select **Import**. You will need to import the AWS recommended templates into your template group.
3. Use the **CloudWatch alarm template groups** dropdown to select an AWS recommended group. These groups contain curated alarms for specific services.
4. Select the templates to import using the check boxes. Each template will list its metrics, preconfigured monitoring values, and provide a description of the metric. When you are done selecting templates, select the **Add** button.

5. The selected templates will move to the **Alarm template(s) to import** section. Review your choices and select **Import**.
6. After the import is complete, the selected templates will be added to the template group. If you want to add more templates, repeat the import process.
7. Imported templates can be customized after import. Alarm settings can be modified to fit your alarming needs.

EventBridge rule groups and templates for monitoring your AWS media workflow

CloudWatch uses Amazon EventBridge rules to send notifications. You begin by creating an event template group. In that event template group, you create event templates that determine what conditions create a notification and who is notified.

This section covers the creation of EventBridge rules using workflow monitor. For more information about how the EventBridge service uses rules, see: [EventBridge rules](#) in the *Amazon EventBridge User Guide*

Creating event template groups

Event template groups allow you to sort and classify events based on your use case.

To create an event template group

1. From the workflow monitor console's navigation pane, select **EventBridge rule templates**.
2. Select **Create event template group**.
3. Give the alarm template group a unique **Group name** and optional **Description**.
4. Select **Create**, You will be taken to the newly created alarm template group's details page.

Creating event templates

You can send notifications based on event templates you create.

To create an event template

1. From the event template group's details page, select **Create event template**.
2. Give the event template a unique **Template name** and optional **Description**.
3. In the **Rule settings** section:

1. Select an **Event type**. When selecting an event type, you can choose between several events created by AWS or select **Signal map active alarm** to use an alarm created by an alarm template.
2. Select a **Target service**. This determines how you would like to be notified of this event. You can select Amazon Simple Notification Service or CloudWatch logs.
3. After selecting a target service, select a **Target**. This will be a Amazon SNS topic or a CloudWatch log group, depending on your target service selection.
4. Select **Create** to complete the process.

Workflow monitor signal maps

Signal maps are visual mappings of AWS resources in your media workflow. You can use workflow monitor to start the signal map discovery on any of the supported resource types. During the discovery process, workflow monitor will automatically and recursively map all connected AWS resources. After the signal map has been created, you can use the workflow monitor console to do things like deploy monitoring templates, view metrics, and view details of the mapped resources.

Topics

- [Creating signal maps for AWS media workflows](#)
- [Viewing signal maps of AWS media workflows](#)
- [Attaching alarm and event templates to the signal map of your AWS media workflow](#)
- [Deploying templates to the signal map of your AWS media workflow](#)
- [Updating the signal map of your AWS media workflow](#)
- [Deleting the signal map of your AWS media workflow](#)

Creating signal maps for AWS media workflows

You can use workflow monitor signal maps to create a visual mapping of all connected AWS resources in your media workflow.

To create a signal map

1. From the workflow monitor console's navigation pane, select **Signal maps**.
2. Select **Create signal map**.

3. Give the signal map a **Name** and **Description**.
4. In the **Discover new signal map** section, resources in the current account and selected region are displayed. Select a resource to begin signal map discovery. The selected resource will be the starting point for discovery.
5. Select **Create**. Allow a few moments for the discovery process to complete. After the process is complete, you will be presented with the new signal map.

 Note

Previews generated in the workflow monitor signal map for AWS Elemental MediaPackage channels are delivered from the MediaPackage Origin Endpoint and will incur Data Transfer Out charges. For pricing, see: [MediaPackage pricing](#).

Viewing signal maps of AWS media workflows

Workflow monitor signal maps allow you to see a visual mapping of all connected AWS resources in your media workflow.

Signal map views

After selecting a signal map, you have two views that can be used to monitor or configure the signal map. **Monitor signal map** and **Configure signal map** is a context-sensitive button found in the upper-right of the signal map console section.

If you select the signal map using the **Signal maps** section of the navigation pane, your signal map will be displayed in the configuration view. The configuration view allows you to make changes to the template groups attached to this signal map, deploy the attached templates, and view the basic details and tags of the signal map.

If you select the signal map using the **Overview** section of the navigation pane, your signal map will be displayed in monitoring view. The monitoring view displays the CloudWatch alarms, EventBridge rules, alerts, logs, and metrics for this signal map.

The view can be changed at any time by selecting the **Monitor/Configure signal map** button in the upper-right. The configuration view requires administrator-level IAM permissions. Required IAM permissions can be viewed here: [Workflow monitor IAM policies](#)

Navigating the signal map

A signal map will contain nodes for every supported AWS resource discovered by workflow monitor. Certain resources, such as MediaLive channels and MediaPackage endpoints can display thumbnail previews of the content, if thumbnail previews are available.

Selecting a resource node, and selecting **View selected resource details** from the **Actions** dropdown menu will take you to the associated service's details page. For example, selecting a MediaLive channel and selecting **View selected resource details** will open the MediaLive console's details page for that channel.

Selecting a resource node will filter the list of active alarms to only that node. If you select the resource's **Target ARN** in the active alarm, you will be taken to the associated service's details page, with the selected resource open.

Attaching alarm and event templates to the signal map of your AWS media workflow

After you have created alarm and event templates, you need to attach these to a signal map. Any of the alarm and event templates you have created can be attached to any discovered signal maps.

To attach alarm and event templates to your signal map

1. From the workflow monitor console's navigation pane, select **Signal maps** and select the signal map you want to work with.
2. In the upper-right of the signal map page, in the **CloudWatch alarm template groups** tab, select **Attach CloudWatch alarm template groups**.
 1. In the new section that opens, choose all of the alarm template groups that you want to apply to this signal map, then select **Add**. This will cause the selected alarm template groups to move to the **Attached CloudWatch alarm template groups** section.
 2. Selecting **Save** will save your changes and return you to the signal map page.
3. At the right of the signal map page, select the **EventBridge rule template groups** tab then select **Attach EventBridge rule template groups**.
 1. In the new section that opens, choose all of the event template groups that you want to apply to this signal map, then select **Add**. This will cause the selected rule template groups to move to the **Attached EventBridge rule template groups** section.
 2. Selecting **Save** will save your changes and return you to the signal map page.
4. You have assigned CloudWatch alarm and EventBridge rule templates to the signal map, but the monitoring is not yet deployed. The next section will cover the deployment of the monitoring resources.

Deploying templates to the signal map of your AWS media workflow

After you have attached the alarm and event templates to your signal map, you must deploy the monitoring. Until the deployment is complete, the monitoring of your signal map will not be active.

Workflow monitor will only deploy alarms that are relevant to the selected signal map. For example, the attached alarm template group might contain alarms for multiple services, such as MediaLive, MediaPackage, and MediaConnect. If the selected signal map only contains MediaLive resources, no MediaPackage or MediaConnect alarms will be deployed.

To deploy the monitoring templates

1. After attaching alarm and event template groups to your signal map and saving your changes, select **Deploy monitor** in the **Actions** dropdown menu.
2. You will be asked to confirm the deployment and presented with the number of CloudWatch and EventBridge resources that will be created. If you would like to proceed, select **Deploy**.

Note

There is no direct cost for using workflow monitor. However, there are costs associated with the resources created and used to monitor your workflow.

When monitoring is deployed, Amazon CloudWatch and Amazon EventBridge resources are created. When using the AWS Management Console, prior to deploying monitoring to a signal map, you will be notified of how many resources will be created. For more information about pricing, see: [CloudWatch pricing](#) and [EventBridge pricing](#). Workflow monitor uses AWS CloudFormation templates to deploy the CloudWatch and EventBridge resources. These templates are stored in a standard class Amazon Simple Storage Service bucket that is created on your behalf, by workflow monitor, during the deployment process and will incur object storage and recall charges. For more information about pricing, see: [Amazon S3 pricing](#).

3. The status of the deployment is displayed next to the name of the signal map. The deployment status is also visible in the **Stacks** section of the CloudFormation console. After a few moments of resource creation and deployment, your signal map monitoring will begin.

Updating the signal map of your AWS media workflow

If a change is made to your workflow, you might need to rediscover the signal map and redeploy monitoring resources. Workflow monitor is a visualization and monitoring tool that does not have the ability to make any changes to your workflow. Signal maps represent a point-in-time visualization of your workflow. In the event that you add, remove, or significantly modify parts of your media workflow, we recommend that you rediscover the signal map. If you have monitoring resources attached to the signal map, we recommend you redeploy monitoring after the rediscovery process.

To rediscover a signal map

1. From the workflow monitor console's navigation pane, select **Signal maps** and select the signal map you want to work with.
2. Verify that you are in the **Configure signal map** view. For more information about changing views, see: [Viewing signal maps](#)
3. In the upper-right of the signal map page, select the **Actions** dropdown menu. Select **Rediscover**.
4. You will be presented with the rediscovery screen. Select a resource that is a part of the workflow you are rediscovering. Select the **Rediscover** button.
5. The signal map will be rebuilt according to the current workflow. If you need to redeploy monitoring resources, stay on this signal map's page. Any previously attached monitoring templates will remain attached, but will need to be redeployed.

To redeploy monitoring templates after a signal map rediscovery

1. After the rediscovery, you will be directed to the updated signal map. To redeploy the monitoring templates, select **Deploy monitor** from the **Actions** dropdown menu.
2. You will be asked to confirm the deployment and presented with the number of any CloudWatch and EventBridge resources that will be created. If you would like to proceed, select **Deploy**.
3. The status of the deployment is displayed next to the name of the signal map. After a few moments of resource creation and deployment, your signal map monitoring will begin.

Deleting the signal map of your AWS media workflow

If you no longer need a signal map, it can be deleted. If you have monitoring templates deployed on the signal map, the deletion process will ask you to delete any CloudWatch and EventBridge resources that have been deployed to this signal map. Deleting the deployed resources does not affect the templates that created them. This resource deletion is to ensure that you do not have CloudWatch and EventBridge resources that are deployed but not used.

To delete a signal map

1. From the workflow monitor console's navigation pane, select **Signal maps** and select the radio button next to the signal map you want to delete.
2. Select the **Delete** button. You will be asked to confirm the deletion of the monitoring resources. Select **Delete** to begin the monitoring resource deletion process.
3. The **Monitor deployment** column will display the current status. When the status has changed to **DELETE_COMPLETE**, select the **Delete** button again.
4. You will be asked to confirm deletion of the signal map. Select **Delete** to proceed and delete the signal map.

Workflow monitor quotas

The following section contains quotas for workflow monitor resources. Each quota is on a "per account" basis. If you need to increase a quota for your account, you can use the [AWS Service Quotas console](#) to request an increase, unless otherwise noted in the following table.

Quotas

Resource type	Quota
CloudWatch alarm template groups	20
CloudWatch alarm templates	200
EventBridge rule template groups	20
EventBridge rule templates	200
Signal maps	30

Resource type	Quota
Signal maps: CloudWatch alarm template groups attached to a single signal map	5 You cannot increase this quota.
Signal maps: EventBridge rule template groups attached to a single signal map	5 You cannot increase this quota.

Using workflow monitor

Use the **overview** and **signal maps** sections of the workflow monitor console to review the current status of the workflows and any associated alarms, metrics, and logs.

Topics

- [Workflow monitor overview](#)
- [Overview logs and metrics for workflow monitor](#)
- [Using workflow monitor signal maps](#)

Workflow monitor overview

The **Overview** section of the workflow monitor console is a dashboard that provides at-a-glance information about your signal maps. In the overview section, you can see the current state of each signal map's monitoring, as well as CloudWatch metrics and any associated CloudWatch logs. You can select any signal map to be taken to that signal maps console page.

Overview filtering

Using the **Search** bar in the overview section, you can filter the list of signal maps using context sensitive constraints. After selecting the search bar, you will be presented with a list of **Properties** to filter by. Selecting a property will present **Operators** such as Equals, Contains, Does not equal, and Does not contain. Selecting an operator will create a list of resources from the selected property type. Selecting one of these resources will cause the signal map list to only display signal maps that fit the constraint you defined.

Overview logs and metrics for workflow monitor

To view CloudWatch metrics and logs for a signal map, select the radio button next to the name of the signal map. A tabbed interface for both metrics and logs will appear beneath the signal map list.

CloudWatch Metrics

CloudWatch metrics for the selected signal map will be context-sensitive and only display metrics associated with the services used in that signal maps workflow. You can use the on-screen metrics tools to customize the displayed metric periods and time ranges.

CloudWatch Logs

If you associated a CloudWatch log group with the signal map, that group will be displayed here.

Using workflow monitor signal maps

From the **overview** section of the console, you can select a specific signal map to view more information about that signal map and its attached monitoring resources.

After selecting a signal map, you will be presented with the signal map and a number of tabbed section containing more information:

- CloudWatch alarms
- EventBridge rules
- AWS Elemental alerts
- Metrics
- Logs
- Basic details

Navigating the signal map

A signal map will contain nodes for every supported AWS resource discovered by workflow monitor. Certain resources, such as MediaLive channels and MediaPackage endpoints can display thumbnail previews of the content, if thumbnail previews are available.

Selecting a resource node, and selecting **View selected resource details** from the **Actions** dropdown menu will take you to the associated service's details page. For example, selecting a

MediaLive channel and selecting **View selected resource details** will open the MediaLive console's details page for that channel.

Selecting a resource node will filter the list of active alarms to only that node. If you select the resource's **Target ARN** in the active alarm, you will be taken to the associated service's details page, with the selected resource open.

Dual-stack (IPv6) endpoints for AWS Elemental MediaTailor

AWS Elemental MediaTailor supports dual-stack endpoints that you can use to connect over either IPv4 or IPv6. These endpoints are functionally equivalent to the standard IPv4-only endpoints. The service behavior is identical, and only the network transport layer differs.

What are dual-stack endpoints?

Dual-stack endpoints advertise both IPv4 and IPv6 addresses. When you use a dual-stack endpoint, DNS resolves to either an IPv4 or IPv6 address based on your network configuration and client preferences. You don't need to change your MediaTailor resources, playback configurations, or ad insertion behavior.

Supported features

The following table lists MediaTailor feature support for dual-stack endpoints.

Feature	Dual-stack support
Control plane APIs (all regions)	Supported
SSAI data plane (session initialization, manifest, segment)	Supported
Channel assembly data plane	Supported
AWS Management Console	Supported
AWS SDK	Coming soon

Dual-stack endpoint format

MediaTailor dual-stack endpoints use the `.api.aws` domain:

- **Control plane:** `mediatailor.region.api.aws`

- **SSAI data plane:** Displayed in the **Dualstack endpoints** section of your playback configuration details in the AWS Management Console.
- **Channel assembly data plane:** Displayed in your channel's output details in the AWS Management Console.

Your existing applications continue to work with the original IPv4-only endpoints (api.mediataylor.*region*.amazonaws.com for control plane, *unique-id*.mediataylor.*region*.amazonaws.com for data plane). For a complete list of available endpoints, see [MediaTailor endpoints and quotas](#) in the *AWS General Reference*.

Using dual-stack endpoints

AWS Management Console

Dual-stack endpoints appear in the **Playback configuration details** page under the **Dualstack endpoints** section. You can copy these URLs directly for use in your player or CDN configuration.

AWS Command Line Interface

Specify the dual-stack endpoint URL when you make API calls:

```
aws mediataylor get-playback-configuration \  
  --name my-config \  
  --endpoint-url https://mediataylor.us-east-1.api.aws
```

Replace *my-config* with the name of your playback configuration and *us-east-1* with your AWS Region.

AWS SDK

Enable dual-stack in your SDK configuration.

Python (boto3):

```
import boto3  
  
client = boto3.client(  
    'mediataylor',
```

```
    endpoint_url='https://mediatailor.us-east-1.api.aws'  
  )
```

JavaScript (AWS SDK v3):

```
import { MediaTailorClient } from "@aws-sdk/client-mediatailor";  
  
const client = new MediaTailorClient({  
    useDualstackEndpoint: true  
});
```

Considerations

Keep the following in mind when you use dual-stack endpoints:

- Dual-stack endpoints are available in all AWS Regions where MediaTailor is available.
- You don't need to change your playback configurations, ad decision server settings, or CDN configuration to use dual-stack endpoints.
- If your CDN uses hostname-pattern-based routing, verify that your rules accommodate the `.api.aws` domain format.
- Standard AWS guidance for dual-stack endpoints applies. For more information, see [Dual-stack and FIPS endpoints](#) in the *AWS General Reference*.

Quotas in AWS Elemental MediaTailor

MediaTailor resources and operations requests are subject to the following quotas (formerly referred to as "limits").

You can use the AWS Service Quotas service to view quotas and request quota increases for MediaTailor, as well as many other AWS services. For more information, see the [Service Quotas User Guide](#).

Quotas on ad insertion

The following table describes the quotas on AWS Elemental MediaTailor ad insertion. Unless otherwise noted, the quotas are not adjustable.

Name	Default quota value	Description
Ad decision server (ADS) length	25,000	The maximum number of characters in an ad decision server (ADS) specification.
Ad decision server (ADS) redirects	7	The maximum depth of redirects that MediaTailor follows in VAST wrapper tags. MediaTailor gives up if there are additional redirects.
Ad decision server (ADS) timeout	3	The maximum number of seconds that MediaTailor waits before timing out on an open connection to an ad decision server (ADS).

Name	Default quota value	Description	
		When a connection times out due to no response from the ADS, MediaTailor is unable to fill the ad avail with ads.	
Ad Insertion Requests	10,000	The maximum requests per second to make for personalized manifests when performing server side ad insertion. Ad insertion handles incoming requests for manifests, session initialization, tracking data, and ad segments. This quota is adjustable .	
Configurations	1,000	The maximum number of configurations that MediaTailor allows.	
Content origin length	512	The maximum number of characters in a content origin specification.	

Name	Default quota value	Description	
Content origin server timeout	2	The maximum number of seconds that MediaTailor waits before timing out on an open connection to the content origin server when requesting template manifests. Timeouts generate HTTP 504 (GatewayTimeoutException) response errors.	
Manifest size	2	The maximum size, in MB, of any origin playback manifest. To ensure that you stay under the quota, use gzip to compress your input manifests into MediaTailor.	
Package configurations	5	The maximum number of package configurations per source (whether live or video on demand).	

Name	Default quota value	Description	
Prefetch schedules	25	The maximum number of active prefetch schedules per playback configuration. Expired prefetch schedules don't count towards this limit.	
Server side reporting beacon request timeout	3 seconds	The maximum number of seconds that MediaTailor waits before timing out on an open connection to the server when firing a beacon for server side reporting. When a connection times out, MediaTailor is unable to fire the beacon and the service logs an <code>ERROR_FIRING_BEACON_FAILED</code> message to the <i>MediaTailor/AdDecisionServer/Interactions</i> log in CloudWatch.	

Name	Default quota value	Description	
PRE_SESSION_INITIALIZATION hook timeout	2 seconds	The maximum number of seconds that a PRE_SESSION_INITIALIZATION hook execution can take, including all function steps and HTTP calls. If the hook exceeds its timeout, MediaTailor discards all output and proceeds as if no function was executed.	
PRE_ADS_REQUEST hook timeout	2 seconds	The maximum number of seconds that a PRE_ADS_REQUEST hook execution can take, including all function steps and HTTP calls. If the hook exceeds its timeout, MediaTailor discards all output and proceeds as if no function was executed.	
Maximum functions per hook	20	The maximum number of total function executions allowed per hook.	

Name	Default quota value	Description	
Session expiration	10 times the manifest duration	The maximum amount of time that MediaTailor allows a session to remain inactive before ending the session. Session activity can be a player request or an advance by the origin server. When the session expires, MediaTailor returns an HTTP 400 (Bad Request) response error.	

Quotas on channel assembly

The following table describes the quotas on AWS Elemental MediaTailor channel assembly. Unless otherwise noted, the quotas are [adjustable](#).

Name	Default quota value	Description	
Channel manifest requests, per account	400	The maximum number of egress manifest requests per second for all Channel Assembly channels on an account.	
Channel manifest requests, per channel	50	The maximum number of egress manifest requests per	

Name	Default quota value	Description	
		second for any single Channel Assembly channel.	
Channel outputs	5	The maximum number of outputs per channel.	
Channels per account	100	The maximum number of channels per account.	
Live sources	50	The maximum number of live sources for the source location.	
Programs per channel	400	The maximum number of programs per channel.	
Segment delivery configurations	5	The maximum number of segment delivery configurations per source location.	
Source locations	50	The maximum number of source locations per account.	
VOD Sources	1,000	The maximum number of video on demand (VOD) sources for the source location.	

The following table describes the throttling limits on AWS Elemental MediaTailor channel assembly. Unless otherwise noted, the quotas are [adjustable](#).

Name	Default transactions-per-second maximum limit	Description	
ConfigureLogsForChannel	1	Configure logs for the channel.	
CreateChannel	1	Create a channel.	
CreateLiveSource	1	Create a live source.	
CreateProgram	3	Create a program.	
CreateSourceLocation	1	Create a Source Location.	
CreateVodSource	1	Create a VOD source.	
DeleteChannel	1	Delete a channel.	
DeleteChannelPolicy	1	Delete a channel policy.	
DeleteLiveSource	1	Delete a live source.	
DeleteProgram	3	Delete a program.	
DeleteSourceLocation	1	Delete a source location.	
DeleteVodSource	1	Delete a VOD source.	
DescribeChannel	5	Describe a channel.	
DescribeLiveSource	5	Describe a live source.	
DescribeProgram	5	Describe a program.	

Name	Default transactions-per-second maximum limit	Description	
DescribeSourceLocation	5	Describe a source location.	
DescribeVodSource	5	Describe a VOD source.	
GetChannelPolicy	5	Get a channel policy.	
GetChannelSchedule	5	Get a channel schedule.	
ListAlerts	5	List alerts.	
ListChannels	5	List channels.	
ListLiveSources	5	List live sources.	
ListPrograms	5	List programs.	
ListSourceLocations	5	List source locations.	
ListTagsForResource	5	List tags for a resource.	
ListVodSources	5	List VOD sources.	
PutChannelPolicy	3	Put a channel policy.	
StartChannel	1	Start a channel.	
StopChannel	1	Stop a channel.	
TagResource	1	Tag a resource.	
UntagResource	1	Untag a resource.	
UpdateChannel	1	Update a channel.	

Name	Default transactions-per-second maximum limit	Description	
UpdateLiveSource	1	Update a live source.	
UpdateProgram	1	Update a program.	
UpdateSourceLocation	1	Update a source location.	
UpdateVodSource	1	Update a VOD source.	

Troubleshooting MediaTailor

If you encounter playback errors or similar difficulties when working with AWS Elemental MediaTailor, consult the topics in this section.

Topics

- [Troubleshooting MediaTailor event flow issues](#)
- [Troubleshooting playback from MediaTailor](#)
- [Beacon deduplication](#)
- [Parameter troubleshooting](#)
- [MediaTailor ad skipping troubleshooting guide](#)

Troubleshooting MediaTailor event flow issues

Understanding AWS Elemental MediaTailor event flow provides a powerful foundation for troubleshooting ad insertion issues. By analyzing the sequence, timing, and patterns of events, you can quickly identify where problems occur and implement targeted solutions.

This section provides practical guidance for using event flow analysis to diagnose issues. For understanding the basic event flow concepts, see [Ad insertion event flow](#).

Identifying incomplete event flows

Incomplete event flows occur when the expected sequence of events stops before reaching successful manifest personalization (the process of MediaTailor inserting personalized ad information into the manifest). Identifying where the flow breaks helps pinpoint the root cause of ad insertion failures.

Common incomplete flow patterns

Different failure points in the event flow indicate specific types of problems, such as the following.

- **Flow stops after ad opportunity detection:** Indicates issues with ad markers or the manifest itself that prevent MediaTailor from making an ADS request. ADS connectivity, configuration, or timeout problems would occur after the ADS request is made.

- **Flow stops after ADS request:** Suggests ADS response issues, VAST parsing problems, creative processing failures, ADS timeouts, connectivity errors, or configuration issues such as invalid ADS URLs that are only discovered when the request is made.
- **Missing tracking beacon:** Might indicate tracking configuration issues, server-side reporting problems, or client-side implementation gaps.

CloudWatch queries for incomplete flow analysis

Use these Amazon CloudWatch Logs Insights queries to identify incomplete event flows. Run these queries against the appropriate log groups based on the type of analysis needed.

Log group selection:

- **MediaTailor/AdDecisionServerInteractions** - Use for queries analyzing ad decision server interactions, ad opportunities, and ADS-related failures.
- **MediaTailor/TranscodeService** - Use for analyzing issues where ads were not inserted due to transcoding problems, creative processing failures, or other non-ADS related issues.

Example identify ad opportunities without successful manifest personalization

Log group: MediaTailor/AdDecisionServerInteractions

The following query identifies ad opportunities that did not result in successful manifest personalization:

```
fields @timestamp, eventType, avail.availId, sessionId
| filter eventType = "AD_MARKER_FOUND"
| stats count() as total_opportunities by avail.availId
| join (
  fields @timestamp, eventType, avail.availId
  | filter eventType = "FILLED_AVAIL"
  | stats count() as successful_fills by avail.availId
) on avail.availId
| where ispresent(total_opportunities) and not ispresent(successful_fills)
| sort total_opportunities desc
```

Example analyze event flow completion rates

Log group: MediaTailor/AdDecisionServerInteractions

The following query analyzes completion rates across different event types:

```
fields @timestamp, eventType, avail.availId
| filter eventType in ["AD_MARKER_FOUND", "MAKING_ADS_REQUEST", "VAST_RESPONSE",
  "FILLED_AVAIL", "BEACON_FIRED"]
| stats count() by eventType, avail.availId
| sort avail.availId, eventType
```

Example find sessions with missing beacon events

Log group: MediaTailor/AdDecisionServerInteractions

The following query identifies sessions that have filled avails but no corresponding beacon events:

```
fields @timestamp, eventType, sessionId, avail.availId
| filter eventType = "FILLED_AVAIL"
| stats count() as filled_avails by sessionId
| join (
  fields @timestamp, eventType, sessionId
  | filter eventType = "BEACON_FIRED"
  | stats count() as beacon_events by sessionId
) on sessionId
| where filled_avails > 0 and (not ispresent(beacon_events) or beacon_events = 0)
| sort filled_avails desc
```

Example identify transcoding-related ad insertion failures

Log group: MediaTailor/TranscodeService

The following query identifies transcoding issues that prevent successful ad insertion:

```
fields @timestamp, eventType, sessionId, requestId
| filter eventType in ["TRANSCODE_IN_PROGRESS", "INTERNAL_ERROR", "MISSING_VARIANTS",
  "PROFILE_NOT_FOUND"]
| stats count() as transcode_issues by eventType, sessionId
| sort transcode_issues desc
```

Analyzing event timing issues

Event timing analysis helps identify performance bottlenecks and optimize ad insertion workflows. Unusual timing patterns often indicate underlying issues that affect viewer experience.

Performance timing thresholds

Use these timing thresholds to identify potential performance issues.

- **Total flow duration more than 5 seconds:** Can impact viewer experience and can indicate ADS performance issues, origin server problems (such as manifest retrieval timeouts), or internal MediaTailor issues including infrastructure problems with NAT Gateway, DynamoDB, EC2, or other system components.
- **ADS response time more than 2 seconds:** Suggests ADS performance problems or network latency issues.
- **Manifest personalization more than 1 second:** Can indicate creative processing delays, origin server issues (such as manifest retrieval timeouts), or internal MediaTailor system problems including infrastructure constraints with NAT Gateway, DynamoDB, EC2, or other components.

Timing analysis queries

Use these queries to analyze event timing patterns.

Example measure total event flow duration

The following query measures the total duration of event flows and identifies those exceeding 5 seconds:

```
fields @timestamp, eventType, avail.availId
| filter avail.availId = "your-avail-id"
| filter eventType in ["AD_MARKER_FOUND", "FILLED_AVAIL"]
| sort @timestamp asc
| stats min(@timestamp) as start_time, max(@timestamp) as end_time by avail.availId
| eval duration_seconds = (end_time - start_time) / 1000
| where duration_seconds > 5
```

Example analyze ADS response timing

The following query analyzes ADS response times and identifies those exceeding 2 seconds:

```
fields @timestamp, eventType, avail.availId
| filter avail.availId = "your-avail-id"
| filter eventType in ["MAKING_ADS_REQUEST", "VAST_RESPONSE"]
| sort @timestamp asc
```

```
| stats min(@timestamp) as request_time, max(@timestamp) as response_time by  
  avail.availId  
| eval ads_response_seconds = (response_time - request_time) / 1000  
| where ads_response_seconds > 2
```

Example identify slow manifest personalization

The following query identifies manifest personalization processes that take longer than 1 second:

```
fields @timestamp, eventType, avail.availId  
| filter avail.availId = "your-avail-id"  
| filter eventType in ["VAST_RESPONSE", "FILLED_AVAIL"]  
| sort @timestamp asc  
| stats min(@timestamp) as response_time, max(@timestamp) as filled_time by  
  avail.availId  
| eval personalization_seconds = (filled_time - response_time) / 1000  
| where personalization_seconds > 1
```

Common event flow problems and solutions

This section provides solutions for frequently encountered event flow issues, organized by problem type and symptoms.

Ad decision server request failures

Symptoms: Event flow stops after ad opportunity detection. No ADS request events logged.

Common causes and solutions

- **ADS URL configuration errors:** Verify the ADS URL in your playback configuration is correct and accessible. In the ads interaction log, you will see an ADS request event (MAKING_ADS_REQUEST) but no corresponding VAST response, often accompanied by an `ERROR_UNKNOWN` or similar error event.
- **Network connectivity issues:** Check network connectivity between MediaTailor and your ADS, including firewall rules and DNS resolution.
- **SSL/TLS certificate problems:** Ensure your ADS uses valid SSL certificates from a trusted certificate authority. For Google Ad Manager specifically, you might need to contact [AWS Support](#) to enable a configuration flag that accepts Google's SSL certificates.

Diagnostic query

The following query helps diagnose ADS request failures by tracking the event sequence:

```
fields @timestamp, eventType, sessionId
| filter sessionId = "your-session-id"
| filter eventType in ["AD_MARKER_FOUND", "MAKING_ADS_REQUEST", "ERROR_ADS_IO",
"ERROR_UNKNOWN_HOST"]
| sort @timestamp asc
```

Ad decision server response failures

Symptoms: ADS requests succeed but MediaTailor doesn't receive a response, or parsing errors occur.

Common causes and solutions

- **Invalid VAST format:** Validate your ADS VAST responses against VAST specification standards.
- **ADS timeout issues:** Increase ADS timeout settings or optimize ADS response time.
- **Empty ad inventory:** Check ad inventory availability and targeting criteria in your ADS configuration.

Diagnostic query

The following query helps diagnose ADS response failures by examining request and response events:

```
fields @timestamp, eventType, sessionId
| filter sessionId = "your-session-id"
| filter eventType in ["MAKING_ADS_REQUEST", "VAST_RESPONSE", "EMPTY_VAST_RESPONSE",
"ERROR_ADS_RESPONSE_PARSE", "ERROR_ADS_TIMEOUT"]
| sort @timestamp asc
```

Manifest personalization failures

Symptoms: VAST responses received but manifest personalization fails or ads are skipped.

Common causes and solutions:

- **Creative transcoding issues:** Check if the ad is a `NEW_CREATIVE`, which requires transcoding prior to insertion. You can also check for transcoding errors by examining the MediaTailor/

TranscodeService log for error events such as INTERNAL_ERROR, MISSING_VARIANTS, or PROFILE_NOT_FOUND.

- **Duration mismatch problems:** Verify ad durations fit within available ad break durations.
- **Personalization threshold issues:** Review personalization threshold settings in your playback configuration.

Diagnostic query

The following query helps diagnose manifest personalization failures by examining VAST responses and filled avails:

```
fields @timestamp, eventType, sessionId, skippedAds
| filter sessionId = "your-session-id"
| filter eventType in ["VAST_RESPONSE", "FILLED_AVAIL", "WARNING_NO_ADVERTISEMENTS"]
| sort @timestamp asc
```

Query for skipped ad reasons

The following query provides detailed information about why ads were skipped:

```
fields @timestamp, eventType, sessionId, skippedAds.reason, skippedAds.creativeUniqueId
| filter sessionId = "your-session-id"
| filter eventType = "WARNING_NO_ADVERTISEMENTS" or ispresent(skippedAds)
| sort @timestamp asc
```

Query for skipped ad reasons and creative unique IDs

The following query provides detailed skipped ad information including reasons and creative unique IDs for the first two ads in each avail:

```
fields @timestamp, eventType
| filter sessionId = "your-session-id"
| filter eventType = "FILLED_AVAIL"
| fields avail.skippedAds.0.vastDuration as SkippedDur_Ad0,
  avail.skippedAds.0.skippedReason as Ad0_SkipReason,
  avail.skippedAds.0.creativeUniqueId as SkippedCreative0_UID
| fields avail.skippedAds.1.vastDuration as SkippedDur_Ad1,
  avail.skippedAds.1.skippedReason as Ad1_SkipReason,
  avail.skippedAds.1.creativeUniqueId as SkippedCreative1_UID
| sort @timestamp desc
```

Tracking beacon failures

Symptoms: Successful manifest personalization but missing or failed tracking beacons.

Common causes and solutions

- **Client-side implementation issues:** Most tracking beacon issues stem from client-side implementation problems, such as not polling tracking URLs frequently enough for client-side tracking, or player-specific beacon firing logic issues.
- **Tracking URL accessibility issues:** Verify that tracking URLs in VAST responses are accessible and return appropriate responses. Issues can occur when URLs are not reachable or when MediaTailor encounters internal issues preventing successful tracking response delivery.
- **Player segment request issues:** Apparent tracking beacon failures can occur when the client player doesn't actually request any segments. This results in no beacons being sent, which appears as a tracking failure but is actually a player implementation issue rather than a beacon problem.

Diagnostic query

The following query helps diagnose tracking beacon failures by examining filled avails and beacon events:

```
fields @timestamp, eventType, sessionId
| filter sessionId = "your-session-id"
| filter eventType in ["FILLED_AVAIL", "BEACON_FIRED", "ERROR_FIRING_BEACON_FAILED"]
| sort @timestamp asc
```

Event flow monitoring best practices

Implement these monitoring practices to proactively identify and resolve event flow issues:

Setting up CloudWatch alarms

Create Amazon CloudWatch alarms to monitor key event flow metrics.

- **Flow completion rate alarm:** Alert when the ratio of successful manifest personalization to ad opportunities drops below acceptable thresholds.
- **ADS response time alarm:** Monitor average ADS response times and alert when they exceed performance thresholds.

- **Error rate alarm:** Track error event frequencies and alert on unusual spikes in specific error types.

Regular monitoring queries

Run these queries regularly to maintain visibility into event flow health:

Example daily event flow success rate

The following query provides a daily overview of event flow success rates by event type:

```
fields @timestamp, eventType
| filter @timestamp > datefloor(@timestamp, 1d)
| stats count() as total_events by eventType
| sort total_events desc
```

Example hourly error rate trending

The following query tracks error rates by hour to identify trending issues:

```
fields @timestamp, eventType
| filter eventType like /ERROR_/
| stats count() as error_count by datefloor(@timestamp, 1h) as hour
| sort hour desc
```

Performance optimization guidance

Use event flow analysis to optimize ad insertion performance.

- **ADS optimization:** Work with your ADS provider to optimize response times and reduce latency.
- **Creative preparation:** Pre-transcode ad creatives to match your content profiles and reduce processing delays.
- **Configuration tuning:** Adjust timeout settings, personalization thresholds, and other configuration parameters based on event flow analysis.

Additional troubleshooting resources

For additional troubleshooting guidance beyond event flow analysis:

- For detailed log format information and technical specifications, see [Viewing logs](#).

- For comprehensive troubleshooting of common ad insertion issues, see [Troubleshooting common issues](#).
- For monitoring and alerting setup guidance, see [Monitoring AWS Elemental MediaTailor with Amazon CloudWatch metrics](#).
- For debug logging procedures, see [Generating debug logs](#).

Troubleshooting playback from MediaTailor

This section provides information about the HTTP error codes that you might receive while testing your player software and during the normal processing of player requests.

Note

You might also receive errors from the AWS Elemental MediaTailor API, during configuration operations like `PutPlaybackConfiguration` and `GetPlaybackConfiguration`. For information about those types of errors, see the [AWS Elemental MediaTailor API Reference](#).

When your player sends a request to AWS Elemental MediaTailor, either directly or through a CDN, MediaTailor responds with a status code. If MediaTailor successfully handles the request, it returns the HTTP status code 200 OK, indicating success, along with the populated manifest. If the request is unsuccessful, MediaTailor returns an HTTP status code, an exception name, and an error message.

AWS Elemental MediaTailor returns two classes of errors:

- **Client errors** – errors that are usually caused by a problem in the request itself, like an improperly formatted request, an invalid parameter, or a bad URL. These errors have an HTTP 4xx response code.
- **Server errors** – errors that are usually caused by a problem with MediaTailor or one of its dependencies, like the ad decision server (ADS) or the origin server. These errors have an HTTP 5xx response code.

Topics

- [Client playback errors returned by AWS Elemental MediaTailor](#)

- [Server playback errors returned by AWS Elemental MediaTailor](#)
- [Playback error examples](#)

Client playback errors returned by AWS Elemental MediaTailor

General guidance:

- You can find detailed information for most errors in the headers and body of the response.
- For some errors, you need to check your configuration settings. You can retrieve the settings for your playback configuration from AWS Elemental MediaTailor. For the API, the resource is `GetPlaybackConfiguration/Name`. For details, see the [AWS Elemental MediaTailor API Reference](#).

The following table lists the client error codes that are returned by the manifest manipulation activities of AWS Elemental MediaTailor, probable causes, and actions you can take to resolve them.

Code	Exception name	Meaning	What to do
400	<code>BadRequestException</code>	MediaTailor is unable to service the request due to one or more errors in formatting or content. A parameter might be improperly formatted, or the request might contain an invalid playback configuration or session ID.	Check that your request is properly formatted and contains accurate information. Make sure that the playback endpoint setting on the player matches the <code>ManifestEndpointPrefix</code> setting returned by <code>GetPlaybackConfiguration</code> . Retry your request.
400	<code>AccessDeniedException</code>	The host header provided in the request doesn't match the manifest endpoint prefix that is configured in the MediaTailor playback URL. Your CDN might be misconfigured.	Check your CDN settings and make sure that you are using the correct manifest endpoint prefix for MediaTailor. Retry your request.

Code	Exception name	Meaning	What to do
404	NotFoundException	MediaTailor is unable to find the information specified. Possible reasons include a URL that doesn't map to anything in the service, a configuration that isn't defined, or a session that is unavailable.	Check your configuration and the validity of your request, and then reinitialize the session.
409	ConflictException	A player tried to load multiple playlists simultaneously for a single session. As a result, MediaTailor detected a session consistency conflict. This problem occurs for HLS players.	Make sure that your player requests playlists one at a time. This is in accordance with the HLS specification.
413	Gone	An AWS Support operator has blocked a player session or customer configuration. AWS Support does this in rare circumstances when we detect a very high volume of 4xx requests coming from errant traffic for a single session or configuration.	If you think that the request shouldn't be blocked, contact AWS Support . They can look into it and remove the blocking filter, if appropriate.

If you need further assistance, contact [AWS Support](#).

Server playback errors returned by AWS Elemental MediaTailor

General guidance:

- You can find detailed information for most errors in the headers and body of the response.
- For some errors, you need to check your configuration settings. You can retrieve the settings for your playback configuration from AWS Elemental MediaTailor. For the API, the resource is `GetPlaybackConfiguration/Name`. For details, see the [AWS Elemental MediaTailor API Reference](#).

The following table lists the server error codes returned by the manifest manipulation activities of AWS Elemental MediaTailor, probable causes, and actions you can take to resolve them.

Code	Exception name	Meaning	What to do
500	InternalServerError	Unhandled exception.	Retry the request. If the problem persists, check the reported health of MediaTailor for your AWS Region at https://status.aws.amazon.com/ .
500	BadGatewayException	Either the origin server address or the ad decision server (ADS) address is invalid. Examples of invalid addresses are a private IP address and localhost .	Make sure that your configuration has the correct settings for your ADS and origin server, and then retry the request.
500	UnsupportedManifestException	Either the origin manifest has changed so that MediaTailor can't personalize it or MediaTailor doesn't support the origin's manifest format.	This might affect only the individual session. Reinitialize the session. You can usually accomplish this by refreshing the page in the viewer. If the problem persists, verify that MediaTailor supports the origin's manifest format. For information, see Integrating a content source .
500	LoadShed	MediaTailor experienced a resource constraint while servicing your request.	Retry the request. If the problem persists, check the reported health of MediaTailor for your AWS Region at https://status.aws.amazon.com/ .
500	ThrottlingException	Your transactions per second have reached your quota, and MediaTailor is throttling your use.	Retry the request. You can also check the reported health of MediaTailor for your AWS Region at https://status.aws.amazon.com/ .

Code	Exception name	Meaning	What to do
			s.amazon.com/ . You might want to increase the quota on your transactions per second. For more information, see the section called “Quotas on ad insertion” .
50	GatewayTimeoutException	A timeout occurred while MediaTailor was contacting the origin server.	Retry the request. If the problem persists, check the health of the origin server and make sure the origin server is responding within the content origin server timeout that is listed at the section called “Quotas on ad insertion” .

If you need further assistance, contact [AWS Support](#).

Playback error examples

This section lists some examples of the playback errors that you might see in command line interactions with AWS Elemental MediaTailor.

The following example shows the result when a timeout occurs between AWS Elemental MediaTailor and either the ad decision server (ADS) or the origin server.

```
~[> curl -vvv https://111122223333444455556666123456789012.mediatailor.us-west-2.amazonaws.com/v1/master/123456789012/Multiperiod_DASH_Demo/index.mpd
* Trying 54.186.133.224...
* Connected to 111122223333444455556666123456789012.mediatailor.us-west-2.amazonaws.com (11.222.333.444) port 555 (#0)
* TLS 1.2 connection using TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
* Server certificate: mediatailor.us-west-2.amazonaws.com
* Server certificate: Amazon
* Server certificate: Amazon Root CA 1
* Server certificate: Starfield Services Root Certificate Authority - G2
> GET /v1/master/123456789012/Multiperiod_DASH_Demo/index.mpd HTTP/1.1
```

```
> Host: 111122223333444455556666123456789012.mediatailor.us-west-2.amazonaws.com
> User-Agent: curl/7.43.0
> Accept: */*
>
< HTTP/1.1 504 Gateway Timeout
< Date: Thu, 29 Nov 2018 18:43:14 GMT
< Content-Type: application/json
< Content-Length: 338
< Connection: keep-alive
< x-amzn-RequestId: 123456789012-123456789012
< x-amzn-ErrorType: GatewayTimeoutException:http://internal.amazon.com/coral/
com.amazon.elemental.midas.mms.coral/
<
* Connection #0 to host 111122223333444455556666123456789012.mediatailor.us-
west-2.amazonaws.com left intact
{"message":"failed to generate manifest: Unable to obtain template playlist.
  origin URL:[https://777788889999.mediapackage.us-west-2.amazonaws.com/out/
v1/444455556666111122223333/index.mpd], asset path: [index.mpd], sessionId:
[123456789012123456789012] customerId:[123456789012]}%
```

Beacon deduplication

This section provides troubleshooting guidance for MediaTailor's beacon deduplication feature in server-side tracking.

Symptom: Lower beacon counts than previously observed

Cause: Beacon deduplication is preventing duplicate events that were previously being counted multiple times.

Resolution: This is expected behavior and indicates more accurate measurement. The new counts represent actual ad viewing events rather than technical segment requests. Update your reporting baselines to reflect the deduplicated counts.

Symptom: Beacon counts vary between different player implementations

Cause: Different players may have varying segment request patterns, but deduplication ensures consistent beacon counts regardless of these differences.

Resolution: This is expected behavior. Deduplication provides consistent measurement across different player types and network conditions.

Symptom: Discrepancy between expected and actual beacon counts

Cause: Previous expectations may have been based on inflated counts due to duplicate beacon firing.

Resolution: Review historical data to identify patterns of duplicate beacons. Use the new deduplicated counts as the accurate baseline for measurement and reporting.

Parameter troubleshooting

For troubleshooting issues related to dynamic ad variables, manifest query parameters, character restrictions, length limitations, and configuration aliases, see [MediaTailor parameter troubleshooting guide](#) in [MediaTailor dynamic ad variables for ADS requests](#).

MediaTailor ad skipping troubleshooting guide

Ad skipping is one of the most common issues that MediaTailor customers report. This guide helps you identify ad skipping issues and provides step-by-step troubleshooting procedures.

Symptoms and impact

When ad skipping occurs, you might observe the following symptoms:

- Ads don't appear during expected ad breaks
- Ad breaks show content instead of advertisements
- Inconsistent ad playback across different viewing sessions
- CloudWatch logs showing AdSkipped events with various skip reasons

Ad skipping directly impacts revenue and can create poor viewer experiences. Address these issues promptly to maintain optimal performance.

Identifying skip reasons

Use CloudWatch Logs Insights to query the MediaTailor/AdDecisionServerInteractions log group for skipped ads:

```
fields @timestamp, avail.availId, skippedAds.0.skippedReason,  
       skippedAds.0.creativeUniqueId  
| filter eventType = "FILLED_Avail" and ispresent(skippedAds.0.skippedReason)  
| sort @timestamp desc
```

This query returns the most recent ad skip events with their specific reasons to help you identify patterns.

Topics

- [MediaTailor NEW_CREATIVE ad skipping troubleshooting](#)
- [MediaTailor ADS timeout ad skipping troubleshooting](#)
- [MediaTailor VAST parsing ad skipping troubleshooting](#)
- [MediaTailor duration mismatch ad skipping troubleshooting](#)
- [MediaTailor session variable ad skipping troubleshooting](#)
- [MediaTailor ad skipping monitoring and alerts](#)
- [MediaTailor ad skipping prevention best practices](#)
- [MediaTailor ad skipping reference guide](#)

MediaTailor NEW_CREATIVE ad skipping troubleshooting

When ads are skipped with the NEW_CREATIVE reason, AWS Elemental MediaTailor encountered an ad that requires transcoding before insertion. This troubleshooting guide explains the causes and provides step-by-step resolution procedures.

What causes NEW_CREATIVE skipping

MediaTailor transcodes ads based on three key factors: creative ID, AWS Account ID, and transcode variant set (the playback renditions for the underlying content stream). When any part of the creative ID or transcode variant set differs, MediaTailor recognizes the ad as a new variant that requires transcoding.

Bitrate changes

Bitrate changes can cause NEW_CREATIVE skipping when the bitrate doesn't match after rounding to the nearest 8,000 bits. This commonly occurs in the following situations:

- The underlying content stream changes the primary manifest

- New sessions are created with different variants than existing sessions
- Content stream bitrates are inconsistent across playback sessions

Harvest job bandwidth variance

For harvest jobs, MediaTailor uses a 15% bandwidth variance threshold when matching ads to content streams. If there is a difference of 15% or more between the bandwidth of the live stream and the bandwidth of the harvested HLS file, the ad will be transcoded again. This can result in `NEW_CREATIVE` skipping during the transcoding process.

This behavior commonly occurs when:

- The same pre-transcoded ads are used for both live streaming and VOD content
- The harvested content has different encoding parameters than the original live stream
- Bandwidth differences exceed the 15% threshold between live and harvested content

Creative ID conflicts

When MediaTailor encounters a different creative ID for a media file that was already transcoded, the following sequence occurs:

1. The ad is skipped with reason `NEW_CREATIVE`
2. This leads to an unnecessary transcoding attempt
3. The creative is marked as `DUPLICATE_TRANSCODE` or `COPY_DEDUP`

Note

MediaTailor does not expire or delete transcoded ads. We store them in a MediaTailor-owned S3 bucket indefinitely.

Resolution steps

To resolve `NEW_CREATIVE` ad skipping issues:

1. Verify that your ad decision server returns consistent creative IDs for the same ad content.
2. Check that your content stream maintains consistent bitrates and variant sets.

3. Consider implementing ad prefetching to ensure ads are transcoded before playback. For more information, see [the section called "Prefetching ads"](#).
4. For persistent issues, contact [AWS Support](#) for additional troubleshooting assistance.

Monitoring NEW_CREATIVE patterns

Use this CloudWatch Logs Insights query to analyze patterns in NEW_CREATIVE ad skipping:

```
fields @timestamp, sessionId, creativeId, skipReason, MediaFileSourceUrl
| filter skipReason = "NEW_CREATIVE"
| stats count() by creativeId, MediaFileSourceUrl
| sort count desc
| limit 50
```

MediaTailor ADS timeout ad skipping troubleshooting

When ads are skipped with ADS_TIMEOUT or related reasons, you have connectivity or performance issues with your ad decision server. AWS Elemental MediaTailor requires reliable communication with your ADS to successfully insert ads. This troubleshooting guide explains how to identify and resolve these connectivity issues.

Common ADS connectivity issues

Common ADS connectivity issues include the following:

- Ad decision server not accessible from MediaTailor
- ADS not responding within the configured timeout period
- ADS unable to handle request volume during peak periods
- Network connectivity issues between MediaTailor and your ADS

Resolution steps

To resolve ADS timeout issues:

1. Verify that your ad decision server is accessible from MediaTailor.
2. Check that your ADS responds within the configured timeout period.

3. Ensure your ADS can handle the request volume during peak periods.
4. Consider implementing a fallback ad strategy for when your primary ADS is unavailable.

Monitoring ADS performance

Set up CloudWatch alarms for the `AdDecisionServer.Timeouts` metric to proactively detect ADS connectivity issues.

MediaTailor VAST parsing ad skipping troubleshooting

When ads are skipped with `VAST_PARSING_ERROR` or `MEDIA_FILE_UNAVAILABLE`, you have issues with your VAST responses or ad media files. AWS Elemental MediaTailor requires properly formatted VAST responses and accessible media files for successful ad insertion. This troubleshooting guide explains how to identify and resolve these issues.

Common VAST issues

Common VAST issues include the following:

- VAST response format not compliant with VAST specification
- Media file URLs in VAST response not publicly accessible
- Special characters not properly encoded in VAST XML
- VAST response missing required media files or formats

Resolution steps

To resolve VAST parsing issues:

1. Validate your VAST response format against the VAST specification.
2. Ensure all media file URLs in the VAST response are publicly accessible.
3. Check for proper encoding of special characters in your VAST XML.
4. Verify that your VAST response includes media files in formats compatible with MediaTailor.

VAST wrapper troubleshooting

For issues with `INVALID_VAST_WRAPPER_AD` or `REJECTED_REPLICA_VAST`:

- Validate VAST wrapper responses against the VAST specification
- Ensure all wrapper elements are properly formatted and contain valid VASTAdTagURI
- Check ad server configuration for duplicate content detection policies
- Ensure VAST responses contain unique creative content within the same session

MediaTailor duration mismatch ad skipping troubleshooting

When ads are skipped with AVAIL_DURATION_EXCEEDED, LEFTOVER_AVAIL_EXCEEDED_THRESHOLD, or format-related reasons, you have duration or format compatibility issues. AWS Elemental MediaTailor requires proper duration formatting and matching between ad content and available ad break time. This troubleshooting guide explains how to identify and resolve these duration-related issues.

Common duration issues

Common duration issues include the following:

- Ads longer than available ad break duration
- Incorrect ad break markers in content signaling wrong duration
- Personalization threshold not met for ad insertion
- Incorrect duration format in EXT-X-CUE-OUT tags

Duration format requirements

The EXT-X-CUE-OUT tag duration parameter must be formatted as an integer value, not as an ISO 8601 duration format.

Duration format requirements

Format	Example	Status
Integer (Correct)	32	Supported - represents 32 seconds
Decimal (Correct)	30.000	Supported - represents 30 seconds
ISO 8601 (Incorrect)	PT32S	Not supported - causes insertion failures

Resolution steps

To resolve duration mismatch issues:

1. Ensure your ADS returns ads that fit within the available ad break duration.
2. Check that your ad break markers in the content correctly signal the intended duration.
3. Verify that EXT-X-CUE-OUT duration parameters use integer format.
4. Consider adjusting the personalization threshold if appropriate for your use case.

MediaTailor session variable ad skipping troubleshooting

Session variables play a critical role in ad targeting and selection in AWS Elemental MediaTailor. Incorrect session variable configuration is a common cause of ad skipping issues. This comprehensive troubleshooting guide explains how to identify and resolve session variable problems that can prevent successful ad insertion.

Common session variable issues

Common session variable issues include the following:

- **Missing required variables:** Your ad decision server might require specific variables that aren't being provided.
- **Incorrect variable syntax:** Variables must use the correct syntax (for example, `[session.id]` instead of `${session.id}`).
- **URL encoding issues:** Special characters in variable values might need proper URL encoding.
- **Inconsistent player parameters:** Player parameters must be consistently passed across sessions.
- **Dynamic variable resolution failures:** Variables that can't be resolved are replaced with empty strings.
- **SCTE-35 UPID parsing issues:** Problems with segmentation UPID processing can cause session variable resolution failures.

Verifying session variable resolution

To verify that your session variables are being properly resolved:

1. Enable debug logging for your MediaTailor configuration

2. Check the MediaTailor/AdDecisionServerInteractions log group for the actual ADS request URLs
3. Verify that all variables in the template URL have been replaced with appropriate values
4. Look for any variables that were replaced with empty strings, which may indicate resolution failures

SCTE-35 UPID parsing troubleshooting

Problems with SCTE-35 segmentation UPID processing can cause session variable issues:

- **Format requirements:** UPID must have `segmentation_upid_type` of 12 and include `format_identifier` for proper processing
- **Parsing rules:** Decoded UPID can contain colon delimiters for multiple values. The number of template variables and decoded UPID tokens must be equal
- **Invalid formats:** Avoid double colons with no values (e.g., `::` or `:46175218::4053`) as these cause parsing failures

MediaTailor ad skipping monitoring and alerts

Proactive monitoring helps you detect and resolve ad skipping issues before they significantly impact your revenue. AWS Elemental MediaTailor provides comprehensive metrics and logging capabilities that enable effective monitoring of ad insertion performance. This guide explains how to set up effective monitoring for ad skipping issues.

Key CloudWatch metrics to monitor

Set up CloudWatch alarms for these key MediaTailor metrics:

- `AdDecisionServer.Ads.Skipped` - Count of skipped ads
- `AdDecisionServer.Timeouts` - Count of ADS timeouts
- `Avail.FilledDuration` - Duration of filled ad breaks
- `Avail.SlateOnly` - Count of ad breaks filled with slate only

Advanced CloudWatch Logs Insights queries

Use these specialized queries for detailed troubleshooting:

Comprehensive session analysis

For detailed analysis of ad insertion behavior for a specific session:

```
fields @timestamp, sessionId, eventType, creativeId, skipReason, adBreakIndex
| filter sessionId = "your-session-id-here"
| filter eventType in ["FILLED_AVAIL", "SKIPPED_AVAIL", "MAKING_ADS_REQUEST"]
| sort @timestamp asc
| limit 100
```

Finding Creative IDs

To identify Creative IDs from FILLED_AVAIL events:

```
fields @timestamp, sessionId, eventType
| filter sessionId like /sessionId/ and eventType!='BEACON_FIRED'
| sort @timestamp desc
```

Note

Replace sessionId with the actual session ID you're investigating.

MediaTailor ad skipping prevention best practices

Implementing these best practices helps prevent ad skipping issues before they occur, ensuring better ad insertion performance and revenue protection with AWS Elemental MediaTailor. These proactive measures address the most common causes of ad skipping and help maintain consistent ad delivery.

Proactive measures

- **Implement ad prefetching:** Use MediaTailor's prefetch feature to ensure ads are transcoded before playback. See [the section called “Prefetching ads”](#) for implementation details
- **Maintain consistent creative IDs:** Ensure your ad decision server uses consistent creative IDs for the same ad content across sessions

- **Ensure proper duration formatting:** Use integer values for EXT-X-CUE-OUT duration parameters instead of ISO 8601 format
- **Configure VOD optimization:** Set the **Maximum concurrent ADS requests** setting for VOD streams with multiple ad breaks to reduce server load. For more information, see [Advanced settings](#)
- **Optimize ADS performance:** Configure your ad decision server to respond quickly and handle peak traffic volumes

Implementation guidelines

- **Implement proper error handling:** Configure slate content to fill ad breaks when ads cannot be inserted
- **Test thoroughly:** Validate your ad insertion workflow across different devices and network conditions
- **Implement fallback strategies:** Configure backup ad sources or default ads for when primary ad sources fail
- **Monitor transcoding patterns:** Monitor CloudWatch logs for transcoding efficiency and contact [AWS Support](#) if you notice patterns that indicate transcoding issues

MediaTailor ad skipping reference guide

This section provides comprehensive reference information about ad skip reasons and links to related documentation for AWS Elemental MediaTailor. Use this reference guide to understand the specific meanings of different skip reasons and find additional troubleshooting resources.

Complete ad skip reasons reference

MediaTailor logs specific reasons why ads are skipped in the FILLED_AVAIL event log message from the MediaTailor/AdDecisionServerInteractions log group.

Complete ad skip reasons

Skip reason	Description
NEW_CREATIVE	The ad has not been transcoded yet. This occurs when MediaTailor encounters a new ad creative that requires transcoding before insertion .

Skip reason	Description
PROFILE_NOT_FOUND	The MediaConvert transcode profile associated with the session's configuration does not exist, preventing ad preparation.
TRANSCODE_ERROR	The ad transcode process encountered an error and failed to complete.
TRANSCODE_IN_PROGRESS	The ad transcode is still in progress and not yet ready for insertion.
INTERNAL_ERROR	An internal MediaTailor error occurred while handling the ad, preventing insertion.
AVAIL_DURATION_EXCEEDED	The ad does not fit within the remaining duration of the ad break.
LEFTOVER_AVAIL_EXCEEDED_THRESHOLD	The cumulative duration of all ads that could have been inserted does not meet the personalization threshold configured for the session.
VAST_PARSING_ERROR	The VAST response from the ad decision server contains errors or is malformed.
ADS_TIMEOUT	The ad decision server did not respond within the configured timeout period.
MEDIA_FILE_UNAVAILABLE	The ad media files specified in the VAST response are not accessible.
SESSION_INITIALIZATION_FAILED	The MediaTailor session failed to initialize properly, often due to incorrect session variables.
EARLY_CUE_IN	The ad break ended earlier than expected due to an early cue-in signal, preventing the ad from being fully inserted.
NO_VARIANT_MATCH	The ad creative does not have a variant that matches the content stream's encoding parameters (bitrate, resolution, codec).

Skip reason	Description
NO_MODEL_CREATIVE_MATCH	The ad creative does not match the expected model or format requirements for the current playback configuration.
REJECTED_REPLICA_VAST	The VAST response was rejected due to replica or duplicate content detection policies.
INVALID_VAST_WRAPPER_AD	The VAST wrapper ad contains invalid or malformed wrapper elements that prevent successful ad insertion.
IMPORT_ERROR	An error occurred during the ad import process, preventing the ad from being processed for insertion.
IMPORT_IN_PROGRESS	The ad import process is currently in progress and has not completed yet.

Related resources

For more information on troubleshooting ad skipping issues, refer to these related topics:

- [the section called “Dynamic ad variables”](#) - Comprehensive guide to dynamic ad variables in MediaTailor
- [the section called “Prefetching ads”](#) - How to implement ad prefetching to prevent transcoding-related skipping
- [the section called “Monitoring with CloudWatch metrics”](#) - Monitoring MediaTailor with CloudWatch metrics
- [the section called “Viewing logs”](#) - How to view and analyze MediaTailor logs
- [the section called “Event flow troubleshooting”](#) - Understanding the ad insertion event flow

AWS Elemental MediaTailor resources

The following table lists related resources that you will find useful as you work with AWS Elemental MediaTailor.

Resource	Description
SCTE standard: SCTE 35	The SCTE standard document for SCTE35.
Classes and workshops	Links to role-based and specialty courses as well as self-paced labs to help sharpen your AWS skills and gain practical experience.
AWS developer tools	Links to developer tools, SDKs, IDE tool kits, and command line tools for developing and managing AWS applications.
AWS whitepapers	Links to a comprehensive list of technical AWS whitepapers, covering topics such as architecture, security, and economics and authored by AWS Solutions Architects or other technical experts.
AWS Support center	The hub for creating and managing your AWS Support cases. Also includes links to other helpful resources, such as forums, technical FAQs, service health status, and AWS Trusted Advisor.
AWS Support	The primary webpage for information about AWS Support, a one-on-one, fast-response support channel to help you build and run applications in the cloud.
Contact Us	A central contact point for inquiries concerning AWS billing, account, events, abuse, and other issues.

Resource	Description
AWS site terms	Detailed information about our copyright and trademark; your account, license, and site access; and other topics.

Document history for AWS Elemental MediaTailor

The following table describes important changes to this documentation.

Change	Description	Date
Added ADS timeout, personalization time budget, and concurrency settings	Added documentation for configurable ad decision server (ADS) request timeouts, personalization time budgets, and concurrency on playback configurations, including new prefetch timeout settings.	July 23, 2026
Account-level dimensions for skipped ad reasons	Added account-level (no dimensions) emissions for the SkippedReason metrics.	July 21, 2026
New metric and account-level dimensions	Added the Avail.Complete metric. Added account-level (no dimensions) emissions for the Requests, AdsBilled, Avail.Impression, Avail.Complete, Avail.FilledDuration, Avail.Duration, and AdDecisionServer.Ads metrics.	June 30, 2026
Automatic server-side authentication with Google's ad platforms	Updated the chapter to reflect that MediaTailor now automatically authenticates server-to-server connections with Google Ad Manager (GAM), Google Campaign Manager (GCM), and Google	April 15, 2026

Display & Video 360 (DV360).
No customer action or support ticket is required.
Updated the server-side section to describe the URI-based rules MediaTailor applies and removed the manual support ticket procedure.

[Added TPS traffic shaping](#)

Updated sections to include information about limiting requests to the ADS based on TPS.

October 2, 2025

[Added prefetchScheduleName](#)

Added information about prefetchScheduleName in the ADS logs.

September 24, 2025

[Restructured ad skipping troubleshooting guide](#)

Restructured troubleshooting guide to focus each section on one specific job to be done.

August 26, 2025

[Added query parameter handling documentation](#)

Added section explaining how MediaTailor handles query parameters for origin-specific functionality like time-shifted viewing with MediaPackage.

August 26, 2025

[SGAI VOD configuration guide](#)

Added configuration guidance for server-guided ad insertion with video-on-demand content.

August 7, 2025

[Integrated parameter troubleshooting guidance](#)

Added comprehensive troubleshooting section covering character restriction errors, length limitation issues, configuration alias problems, parameter processing flow debugging, and security best practices.

August 4, 2025

[Enhanced parameter formatting documentation](#)

Added comprehensive character restrictions, URL-encoding support, and length limitations for manifest query parameters and ADS parameters.

August 1, 2025

[Enhanced parameter documentation with comprehensive reference](#)

Added comprehensive parameter character restrictions, length limitations, URL-encoding support, configuration aliases documentation, and session initialization methods based on AWS Guide analysis.

August 1, 2025

[Enhanced manifest query parameters documentation](#)

Added comprehensive parameter placement behavior, character restrictions, URL-encoding support, and protocol-specific differences based on AWS Guide analysis.

August 1, 2025

Enhanced domain variables documentation	Added comprehensive configuration aliases documentation, supported fields, and domain-level parameter rules based on AWS Guide analysis.	August 1, 2025
Added event flow troubleshooting guidance	New section providing practical guidance for using MediaTailor event flow analysis to diagnose and resolve ad insertion issues.	July 31, 2025
Added ad insertion event flow documentation	New section explaining the chronological flow of MediaTailor events during manifest personalization to help users understand the workflow.	July 23, 2025
Enhanced client-side tracking workflow documentation	Added comprehensive client-side reporting workflow documentation with detailed sequence diagram and step-by-step process explanation.	July 16, 2025
CDN integration documentation	Added more comprehensive information about setting up a CDN with MediaTailor.	June 30, 2025
Added CDN integration for Channel Assembly	Added documentation for integrating a CDN with Channel Assembly to optimize content delivery.	May 16, 2025
HLS playlist types	Added definitions for HLS multivariant and media playlists.	May 9, 2025

DASH manifest types	Added definitions for DASH manifest components.	May 9, 2025
Added HLS playlist type information	Added explanations about multivariant and variant playlists to provide context for the examples.	May 9, 2025
Added DASH manifest type information	Added explanations about MPD structure and components to provide context for the examples.	May 9, 2025
Added recurring prefetch	Updated sections to include information about recurring prefetch schedules.	April 28, 2025
Added per-configuration log filtering information	Added information about filtering logs that MediaTailor emits for a configuration.	March 31, 2025
Added vended logs information	Added new section about using vended logs for publishing logs that MediaTailor emits..	February 4, 2025
Added preconditioned ads overview	Added new section that describes how MediaTailor manages ad insertion when you're using preconditioned ads.	January 30, 2025
Added ad conditioning	Added the Streaming media file conditioning setting.	January 30, 2025
SGAI live configuration guide	Added configuration guidance for server-guided ad insertion with live content.	January 15, 2025

Added log types	Added new section that lists the log types that MediaTailor emits.	January 15, 2025
Added log filtering information	Added new section about filtering logs that MediaTailor emits.	January 15, 2025
Added supported query parameter formatting	Added sections for manifest query parameter and ADS query parameter formatting.	January 2, 2025
Integration information for Google Ad Manager	Added a section about integrating MediaTailor with Google Ad Manager from server side and client side.	November 25, 2024
Updated variable	Changed breakabilityStartTime to availabilityStartTime .	May 6, 2024
Added personalization detail	Added the Insertion Mode personalization detail.	May 6, 2024
Program Rules	Added new content on program rules.	April 25, 2024
Updated content on creating channels	Added information about program rules when creating channels.	April 20, 2024
Updated content on adding programs	Added information about program rules when adding programs.	April 20, 2024

Workflow monitor	Analyze AWS media services and create signal maps, visualizations of the media workflow, between those services. Use the signal maps to generate monitoring alarms and notifications using CloudWatch, EventBridge, and CloudFormation.	April 11, 2024
AlternateMedia and the As Run Log	Added a note on how AlternateMedia affects the As Run Log.	February 28, 2024
Updated manifest settings	Added information about passthrough tags, depending on the chosen Ad markup type.	November 28, 2023
SCTE-35 messages for ad breaks	Added information about inserted SCTE-35 tags for Daterange versus Scte35 Enhanced Ad markup types.	November 28, 2023
Key-value pairs for the Enhanced Scte35 Ad markup type	Added information about how MediaTailor handles submitted key-value pairs for the Enhanced Scte35 Ad markup type.	November 28, 2023
VOD source ad opportunities	MediaTailor can now automatically detect ad opportunities on VOD sources.	October 6, 2023
New Autodetect SigV4 authentication type	MediaTailor now supports the AUTODETECT_SIGV4 access type.	August 18, 2023

Updated client-side tracking content	Updated the client-side tracking content to include additional information.	August 12, 2023
Setting up MediaTailor and MediaPackage workflows to use live sources	Added information on setup, general requirements, and behavior when MediaTailor and AWS Elemental MediaPackage workflows use live sources.	May 24, 2023
Manifest query parameters documentation	Added a section that describes manifest query parameters.	April 26, 2023
Overlay ads documentation	Added a section that describes overlay ads.	April 24, 2023
Ad ID decoration in manifests documentation	Added a section that describes ad ID decoration in manifests.	April 24, 2023
Added AFTER_LIVE_EDGE suppression mode	AFTER_LIVE_EDGE ad suppression mode is now available in addition to BEFORE_LIVE_EDGE mode.	February 21, 2023
New As Run log	New topic on the As Run log.	January 19, 2023
IAM best-practices updates	Updated guide to align with the IAM best practices . For more information, see Security best practices in IAM .	December 27, 2022
IAM best-practices updates	Updated guide to align with the IAM best practices . For more information, see Security best practices in IAM .	December 27, 2022

New segmentation UPID private data variables	MediaTailor now supports additional session variables <code>scte.segmentation_upid.private_data.{index}</code> that split colon-delimited private data from segmentation UPID type 12 (MPU).	December 22, 2022
Updated quotas content	Updated and reorganized quotas information.	September 13, 2022
Added and corrected channel assembly quotas	Added quotas for live sources, segment delivery configurations, manifest requests, and channel transactions in the MediaTailor channel assembly service.	September 13, 2022
Added and corrected channel assembly quotas	Added quotas for logging, channel, live sources, programs, source locations , and channel policies in the MediaTailor channel assembly service.	September 11, 2022
New channel assembly alerts table	You can now see a table that explains channel assembly alerts.	September 1, 2022
New Amazon CloudWatch metrics	Added new CloudWatch metrics.	June 26, 2022
Ad calls topic	Added link to prefetch support for VAST responses.	May 25, 2022

[New ADS request variables](#)

MediaTailor now supports these additional SCTE-35 variables in ADS requests:

```
scte.segmentation_
type_id , scte.avai
ls_expected , scte.deli
very_not_restricte
d_flag , scte.segm
ent_num , scte.sub_
segment_num , scte.segm
ents_expected ,
scte.sub_segments_
expected , scte.devi
ce_restrictions ,
scte.no_regional_b
lackout_flag ,
scte.archive_allow
ed_flag and scte.segm
entation_event_id .
```

April 27, 2022

[New IAM managed policy
policy topic](#)

Added two new managed policies for MediaTailor.

November 24, 2021

[New AWSElementalMediaT
ailorReadOnly managed
policy](#)

Added a new AWS managed policy that grants permissions that allows read-only access to MediaTailor resources.

November 10, 2021

[New AWSElementalMediaT
ailorFullAccess
managed policy](#)

Added a new AWS managed policy that allows full access to MediaTailor resources.

November 10, 2021

[New confused deputy topic](#)

Added a topic that explains how to prevent the confused deputy problem.

November 4, 2021

Prefetching ads topic	MediaTailor can now prefetch ads for ad breaks before they occur.	October 12, 2021
Added logging configuration settings for playback configurations	Use logging configuration settings to control settings related to playback configuration logs.	September 28, 2021
SCTE-35 messages for ad breaks	Added information about using segmentation_descriptor messages together with time_signal messages.	September 1, 2021
New linear playback mode	Added a new linear playback mode.	September 1, 2021
New absolute transition type	Added support for absolute transition types, which you can use to set a wall clock start time for your program on linear channels.	September 1, 2021
New channel assembly alerts topic	You can now monitor your channel assembly resources using MediaTailor alerts. When an issue or a potential issue occurs with your channel assembly resources, MediaTailor generates alerts.	July 14, 2021
Corrected channel assembly quotas for channels egress requests	Corrected quotas for channels egress requests in the MediaTailor channel assembly service.	June 29, 2021

New source location authentication type	MediaTailor now supports Secrets Manager access token authentication.	June 16, 2021
New Tier information	Added information about the modes and source types that each Tier supports.	June 13, 2021
New Source type information	For Standard channels, added information about the type of source that the program plays.	June 13, 2021
New MediaTailor live sources documentation	A live source represents a single live stream that you add to your source location. After you create your channel, you can add live sources to your source location and associate each live source with a program.	June 13, 2021
Support for additional UPID types	MediaTailor now supports ADS Information (0xE) and User Defined (0x1) segmentat ion UPID types.	April 15, 2021

[New segmentation UPID dynamic variables](#)

There are three new dynamic variables: `scte.segm entation_upid.asse tId` , `scte.segm entation_upid.cueD ata.key` , and `scte.segm entation_upid.cueD ata.value` . These variables are used in conjuncti on with the MPU segmentat ion UPID type (0xC) for podbuster workflows.

April 15, 2021

[New channel assembly service description](#)

Added information about the new channel assembly service.

March 11, 2021

[New MediaTailor channel assembly service documenta tion](#)

Channel assembly is a new manifest-only service that allows you to create linear streaming channels using your existing video on demand (VOD) content.

March 11, 2021

[Added channel assembly quotas](#)

Added quotas for the new MediaTailor channel assembly service.

March 11, 2021

[New channel assembly terms](#)

Added terms that correspond to the new channel assembly service.

March 10, 2021

Channel assembly tagging support	Added support for tagging of channel assembly resources in AWS Elemental MediaTailor. Channels, SourceLocations, and VodSources support tagging.	March 9, 2021
New dynamic variables topic	MediaTailor now supports dynamic domain variables.	February 25, 2021
Added optional configuration alias settings	Use configuration aliases alongside domain variables to dynamically configure domains during session initialization.	February 25, 2021
New <code>scte.segmentation_upid</code> dynamic ad variable	Added support for the <code>scte.segmentation_upid</code> session data dynamic ad variable.	December 5, 2020
New ad marker passthrough topic	Ad marker passthrough is now available for HLS manifests.	October 29, 2020
Updated configuration advanced settings	Ad marker passthrough is a new playback configuration advanced setting.	October 14, 2020
New debug log mode	New topic on the DEBUG log mode.	August 14, 2020
Clarification around EXT-X-CUE-OUT duration attribute for bumpers	Updated the bumpers requirements so that for HLS the <code>duration</code> attribute is required for each EXT-X-CUE-OUT tag.	August 5, 2020

Session initialization reporting mode parameter	Added support for the <code>reportingMode</code> parameter in session initialization requests. You can now specify whether MediaTailor performs client-side or server-side ad tracking when initializing a session.	July 30, 2020
New bumpers topic	Added a new bumpers topic	July 27, 2020
Ad Suppression is available for DASH	Ad suppression is now available for DASH. Removed the "HLS-only" restriction from the ad suppression topic.	June 3, 2020
Update console-specific names	Updated console-specific names to reflect a newer version of the console UI.	May 1, 2020
New <code>avail.index</code> dynamic ad variable	Added support for the new <code>avail.index</code> session data dynamic ad variable.	March 13, 2020
New <code>AdVerifications</code> and <code>Extensions</code> elements	For client-side reporting, the <code>AdVerifications</code> and <code>Extensions</code> elements are supported.	March 10, 2020
Personalization threshold configuration	Added support for the personalization threshold optional configuration.	February 14, 2020

DASH VOD manifests	Added support for video on demand (VOD) DASH manifests from the origin server, with multi-period manifest output.	December 23, 2019
Console support for transcode profile name	Added description for transcode profile name in the configuration.	December 23, 2019
Updated limits tables	Updated limits for ADS redirects and ADS timeouts.	December 18, 2019
Document live pre-roll behaviors	Added <i>Pre-Roll Ad Insertion</i> section to describe how live pre-roll ads work with AWS Elemental MediaTailor.	November 26, 2019
Support for live pre-roll ads	Added support for inserting pre-roll ads at the beginning of a live stream.	September 11, 2019
Analyzing ADS logs in Amazon CloudWatch Logs insights	Added information for using the AWS Elemental MediaTailor or ADS logs and CloudWatch Logs Insights to analyze your MediaTailor sessions.	August 13, 2019
New security chapter	Added a security chapter to enhance and standardize coverage.	May 23, 2019
DASH single-period manifests	Added support for single-period DASH manifests from the origin server, with multi-period manifest output.	April 4, 2019

Support for SCTE-35 UPIDs in the ADS URL	Added support for including a unique program ID (UPID) in the ad decision server (ADS) URL. This allows the ADS to provide program-level ad targeting within a live linear stream.	March 28, 2019
Client-side reporting supports companion ads	For client-side reporting, the AWS Elemental MediaTailor tracking URL response now includes companion ad metadata.	March 28, 2019
HLS ad marker documentation	Added a section that describes supported HLS ad markers.	March 1, 2019
Tagging support	Added support for tagging of configuration resources in AWS Elemental MediaTailor. Tagging allows you to identify and organize your AWS resources and control access to them and to track your AWS costs.	February 14, 2019
Added AWS CloudTrail logging information	Added topic about using CloudTrail to log actions in the AWS Elemental MediaTailor API.	February 11, 2019

Added section on playback errors	Added information about the errors that might be returned by MediaTailor during playback, in response to requests from a player or a content delivery network (CDN).	February 4, 2019
DASH base64-encoded binary	Added support for providing splicing information in manifests in base64-encoded binary, inside <code><scte35:Signal></code> <code><scte35:Binary></code> markers.	January 4, 2019
DASH time signal	Added support for providing splicing information in manifests inside <code><scte35:TimeSignal></code> markers.	December 5, 2018
DASH location support	Added support for the MPEG-DASH <code><Location></code> tag.	December 4, 2018
DASH support	Added support for MPEG-DASH manifests.	November 14, 2018
Updated limits tables	Updated limits for configurations and manifest size.	October 13, 2018
New and updated metrics	Added metrics for ad decision server (ADS) and origin timeouts, and updated the ADS and origin error definitions to include timed-out responses.	October 13, 2018

Better documentation coverage for server-side and client-side ad insertion use cases	Expanded description and examples to cover the use of dynamic ad variables for server-side ad insertion and for client-side ad insertion.	October 1, 2018
New regions	Added support for the PDX and FRA Regions.	July 18, 2018
VAST/VPAID	Added information about VAST and VPAID.	March 16, 2018
CloudWatch	Added information about available CloudWatch metrics, namespaces, and dimensions.	March 16, 2018
New regions	Added support for the Asia Pacific (Singapore), Asia Pacific (Sydney), and Asia Pacific (Tokyo) Regions.	February 8, 2018
IAM policy information	Added IAM policy information specific to AWS Elemental MediaTailor. Added instructions for creating non-admin roles with limited permissions.	January 3, 2018
First release	First release of this documentation.	November 27, 2017

Note

- The AWS Media Services are not designed or intended for use with applications or in situations requiring fail-safe performance, such as life safety operations, navigation or communication systems, air traffic control, or life support machines in which the

unavailability, interruption or failure of the services could lead to death, personal injury, property damage or environmental damage.