Media connector overview

Table of contents

What is media connector?

The media connector makes media from media sources such as edge attached cameras available to other Azure IoT Operations components.

The media connector can:

- Consume media from endpoints using a variety of protocols, codecs and formats. Supported streaming formats include RTSP and MPEG-TS, and supported file types include AVI, MKV and MP4.
- Publish snapshots to MQTT and save snapshots and video clips to files. It can also stream video to clients and media servers.
- Capture snapshots from a video stream or from an image URL and publish them to an MQTT topic. A subscriber to the MQTT topic can use the captured images for further processing or analysis.
- Save video streams to a local file system on your cluster. Edge Storage Accelerator can provide a reliable and fault-tolerant solution for uploading the captured video to the cloud for storage or processing.
- Proxy a live video stream from a camera to an endpoint that an operator can access. For security and performance reasons, only the media connector should have direct access to an edge camera. The media connector uses a separate media server component to stream video to an operator's endpoint. This media server can transcode to a variety of protocols such as RTSP, RTCP, SRT, and HLS.

Media source	**Example URLs**	**Notes**
Edge attached camera	file://host/dev/video0 file://host/dev/usb0	No authentication required. The URL refers to the device file. Connects to a node using USB, FireWire, MIPI, or proprietary interface.
IP camera	rtsp://192.168.178.45:554/stream1	JPEG over HTTP for snapshots, RTSP/RTCP/RTP/MJPEG-TS for video streams. An IP camera might also expose a standard ONVIF control interface.
Media server	rtsp://192.168.178.45:554/stream1	JPEG over HTTP for snapshots, RTSP/RTCP/RTP/MJPEG-TS for video streams. A media server can also serve images and videos using URLs such

		as ftp://host/path or smb://host/path
Media file	http://camera1/snapshot/profile1 nfs://server/path/file.extension file://localhost/media/path/file.mkv	Any media file with a URL accessible from the cluster.
Media folder	file://host/path/to/folder/ ftp://server/path/to/folder/	A folder, accessible from the cluster, that contain media files such as snapshots or clips.

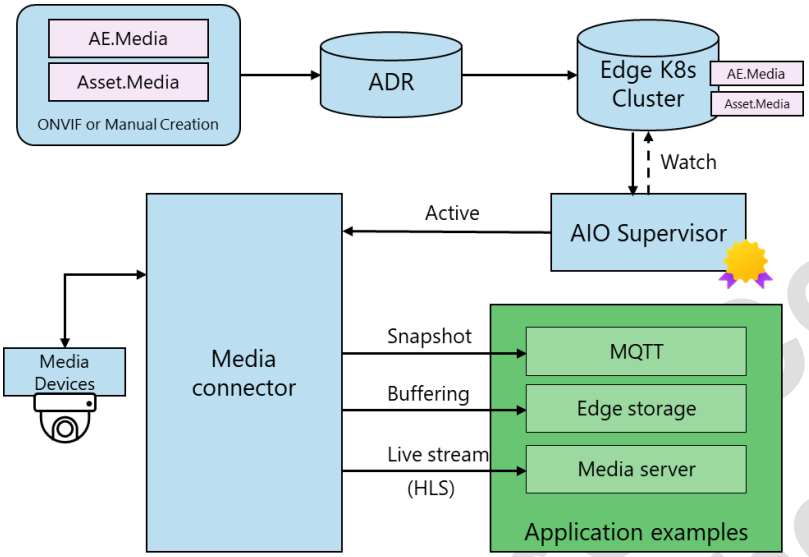
How does it relate to Azure IoT Operations?

The media connector component is part of Azure IoT Operations. The connector deploys to an Arc-enabled Kubernetes cluster on the edge as part of an Azure IoT Operations deployment. The connector interacts with other Azure IoT Operations elements, such as:

- *Asset endpoints*, which are custom resources in your Kubernetes cluster that define connections to resources such as cameras. An asset endpoint configuration includes the URL of the media source, the type of media source, and any credentials that are needed to access the media source. The media connector uses an asset endpoint to access the media source.
- *Assets*, which in Azure IoT Operations Preview are logical entities that you create to represent real assets such as cameras. An Azure IoT Operations camera asset can have properties, tags, and video streams.
- The Azure IoT Operations portal, which is a web UI that provides a unified experience for you to manage assets such as cameras. You can use the portal to configure the assets and asset endpoints that the media connector uses to access media sources.

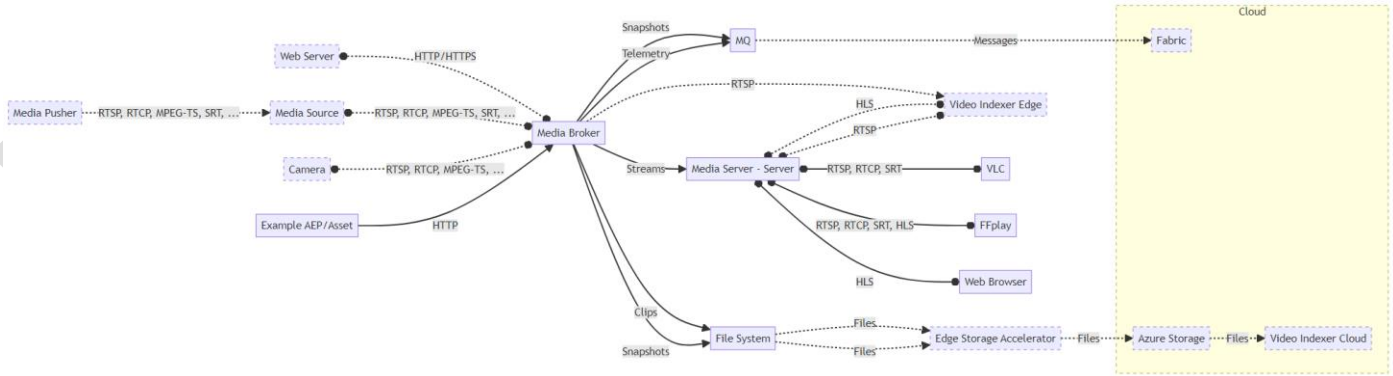
Architecture

The media connector schedules and manages media tasks for media endpoints. The media connector works closely with the ONVIF connector on the RTSP URI, and is compatible with a manually provided URI.



- The AIO supervisor in the Kubernetes cluster reads the asset endpoint custom resource.
- The media connector starts handling the asset related to the endpoint, then starts streaming the media.
- The media connector delivers transcoded media for application use.

The media connector can connect to media endpoints that use a variety of streaming formats such as RTSP and MPEG-TS, or that host files with encodings such as AVI, MKV, and MP4. The media connector can publish snapshots to MQTT or save snapshots and video clips to files. The media connector can also stream video to clients and media servers. The following diagram summarizes the sources and destinations the media connector supports:



The dotted lines represent connections and components that are possible use cases.

API usage

To control the media connector at runtime, you use an mRPC API. mRPC is an MQTTv5 based RPC protocol. It uses MQTT as the transport layer and JSON as the payload. You can use any standard MQTTv5 client to interact with the mRPC API.

For example,

`aio-connectors/asset-operations/<asset-name>/<api-call>`

where `<asset-name>` is the name of the asset and `<api-call>` is `start-task` / `stop-task` / `start-all` / `stop-all`. To start the snapshot-to-mqtt task in the asset:

`aio-connectors/asset-operations/asset-for-example/start-task`

The payload for the API call is JSON, for example, the JSON payload to start the *snapshot-to-fs* task.

Media connector mRPC API usage examples

Pre-requisites

This sample code and script assumes that you have **mosquitto_pub** installed

Initialize variables

Set the development environment variables that this example uses. You can edit the *media-connector-vars.ps1* file (Appendix 2) to customize these values:

```
. ".\media-connector-vars.ps1"
```

Deploy the demo configuration

This demo configuration sets up an asset endpoint and an asset, where .yaml sample files are at Appendix 3 and 4. The AEP points to a public HTTP video stream. The asset references the asset endpoint. The asset configures three data points:

```
kubectl apply -n $aioConnectorsNamespace -f asset endpoint-msft-azure-retail-hls-1.yaml
```

```
kubectl apply -n $aioConnectorsNamespace -f asset-msft-azure-retail-hls-1.yaml
```

Run mRPC API tests on the demo configuration

mRPC is an MQTTv5 based RPC protocol. It uses MQTT as the transport layer and JSON as the payload. You can use any standard MQTTv5 client to interact with the mRPC API.

Set the MQTT broker IP address and port. The default values are for the MQTT broker deployed in the same namespace as the media connector:

```
$mosquittoIngressJson = kubectl get -n aio-connectors services/mosquitto-public -o
jsonpath='{.status.loadBalancer.ingress}'
Write-Host "mosquittoIngressJson:" $mosquittoIngressJson
$mqttIpAddress = ($mosquittoIngressJson | ConvertFrom-Json).ip
if ($mqttIpAddress -eq $null) {
    $mqttIpAddress = ($mosquittoIngressJson | ConvertFrom-Json).hostname
}
```

```
Write-Host "mqttIpAddress:" $mqttIpAddress

$mqttPort=1883
Write-Host "mqttPort:" $mqttPort
```

mRPC ping the endpoint

```
$mrpcTopic = "aio-connectors/endpoint-operations/aep-msft-azure-retail-hls-1/ping"
Write-Host "mrpcTopic:" $mrpcTopic

$mrpcResponseTopic = "$mrpcTopic/response"
Write-Host "mrpcResponseTopic:" $mrpcResponseTopic

$mrpcPayload = "{}"
Write-Host "mrpcPayload:" $mrpcPayload

mosquitto_pub -h $mqttIpAddress -p $mqttPort -q 1 -V mqttv5 -t $mrpcTopic -D PUBLISH
response-topic $mrpcResponseTopic -m $mrpcPayload
```

You can view the ping response on the MQTT topic *aio-brokers/endpoint-operations/aep-msft-azure-retail-hls-1/ping/response*. The response payload is "*pong!*".

> 💡 Use your favorite MQTT client to subscribe to the topic.

Alternatively, you can check the media connector logs to view the ping request and response:

```
kubectl logs -n $aioConnectorsNamespace -l app.kubernetes.io/component=aio-opc-rtsp-1 --
all-containers --ignore-errors --tail=25
```

mRPC ping the asset

```
$mrpcTopic = "aio-connectors/asset-operations/asset-msft-azure-retail-hls-1/ping"
Write-Host "mrpcTopic:" $mrpcTopic

$mrpcResponseTopic = "$mrpcTopic/response"
Write-Host "mrpcResponseTopic:" $mrpcResponseTopic

$mrpcPayload = "{}"
Write-Host "mrpcPayload:" $mrpcPayload
```

```
mosquitto_pub -h $mqttIpAddress -p $mqttPort -q 1 -V mqttv5 -t $mrpcTopic -D PUBLISH
response-topic $mrpcResponseTopic -m $mrpcPayload
```

You can view the ping response on the MQTT *topic aio-brokers/endpoint-operations/asset-msft-azure-retail-hls-1/ping/response*. The response payload is "*pong!*".

> 💡 Use your favorite MQTT client to subscribe to the topic.

Alternatively, you can check the media connector logs to view the ping request and response:

```
kubectl logs -n $aioConnectorsNamespace -l app.kubernetes.io/component=aio-opc-rtsp-1 --
all-containers --ignore-errors --tail=25
```

Control the snapshot-to-mqtt task in the asset

Start the *snapshot-to-mqtt* task in the asset:

```
$mrpcTopic = "aio-connectors/asset-operations/asset-msft-azure-retail-hls-1/start-task"
Write-Host "mrpcTopic:" $mrpcTopic

$mrpcResponseTopic = "$mrpcTopic/response"
Write-Host "mrpcResponseTopic:" $mrpcResponseTopic

$mrpcPayloadFile = "mrpc-payload-start-task-snapshot-to-mqtt.json"
Write-Host "mrpcPayloadFile:" $mrpcPayloadFile

mosquitto_pub -h $mqttIpAddress -p $mqttPort -q 1 -V mqttv5 -t $mrpcTopic -D PUBLISH
response-topic $mrpcResponseTopic -f $mrpcPayloadFile
```

You can check the pod log to see the task status:

```
kubectl logs -n $aioConnectorsNamespace -l app.kubernetes.io/component=aio-opc-rtsp-1 --
all-containers --ignore-errors --tail=25
```

Stop the *snapshot-to-mqtt* task in the asset:

```
$mrpcTopic = "aio-connectors/asset-operations/asset-msft-azure-retail-hls-1/stop-task"
Write-Host "mrpcTopic:" $mrpcTopic

$mrpcResponseTopic = "$mrpcTopic/response"
```

```
Write-Host "mrpcResponseTopic:" $mrpcResponseTopic

$mrpcPayloadFile = "mrpc-payload-stop-task-snapshot-to-mqtt.json"
Write-Host "mrpcPayloadFile:" $mrpcPayloadFile

mosquitto_pub -h $mqttIpAddress -p $mqttPort -q 1 -V mqttv5 -t $mrpcTopic -D PUBLISH
response-topic $mrpcResponseTopic -f $mrpcPayloadFile
```

You can check the pod log to see the task status:

```
kubectl logs -n $aioConnectorsNamespace -l app.kubernetes.io/component=aio-opc-rtsp-1 --
all-containers --ignore-errors --tail=25
```

Control the snapshot-to-fs task in the asset

Start the *snapshot-to-fs* task in the asset:

```
$mrpcTopic = "aio-connectors/asset-operations/asset-msft-azure-retail-hls-1/start-task"
Write-Host "mrpcTopic:" $mrpcTopic

$mrpcResponseTopic = "$mrpcTopic/response"
Write-Host "mrpcResponseTopic:" $mrpcResponseTopic

$mrpcPayloadFile = "mrpc-payload-stop-task-snapshot-to-fs.json"
Write-Host "mrpcPayloadFile:" $mrpcPayloadFile

mosquitto_pub -h $mqttIpAddress -p $mqttPort -q 1 -V mqttv5 -t $mrpcTopic -D PUBLISH
response-topic $mrpcResponseTopic -f $mrpcPayloadFile
```

You can check the pod log to see the task status:

```
kubectl logs -n $aioConnectorsNamespace -l app.kubernetes.io/component=aio-opc-rtsp-1 --
all-containers --ignore-errors --tail=25
```

Stop the *snapshot-to-fs* task in the asset:

```
$mrpcTopic = "aio-connectors/asset-operations/asset-msft-azure-retail-hls-1/stop-task"
Write-Host "mrpcTopic:" $mrpcTopic

$mrpcResponseTopic = "$mrpcTopic/response"
Write-Host "mrpcResponseTopic:" $mrpcResponseTopic
```

```
$mrpcPayloadFile = "mrpc-payload-stop-task-snapshot-to-fs.json"
Write-Host "mrpcPayloadFile:" $mrpcPayloadFile

mosquitto_pub -h $mqttIpAddress -p $mqttPort -q 1 -V mqttv5 -t $mrpcTopic -D PUBLISH
response-topic $mrpcResponseTopic -f $mrpcPayloadFile
```

You can check the pod log to see the task status:

```
kubectl logs -n $aioConnectorsNamespace -l app.kubernetes.io/component=aio-opc-rtsp-1 --
all-containers --ignore-errors --tail=25
```

You can list the files in the pod to view the snapshot files that the *snapshot-to-fs* created:

```
$podName = kubectl get pods -n $aioConnectorsNamespace -l
app.kubernetes.io/component=aio-opc-rtsp-1 --output=jsonpath='{.items[*].metadata.name}'
Write-Host "podName:" $podName
kubectl exec $podName -n $aioConnectorsNamespace -- find /tmp/aio-connectors/data/asset-
msft-azure-retail-hls-1
```

Control the stream-to-rtsp task in the asset

Start the *stream-to-rtsp* task in the asset:

```
$mrpcTopic = "aio-connectors/asset-operations/asset-msft-azure-retail-hls-1/start-task"
Write-Host "mrpcTopic:" $mrpcTopic

$mrpcResponseTopic = "$mrpcTopic/response"
Write-Host "mrpcResponseTopic:" $mrpcResponseTopic

$mrpcPayloadFile = "mrpc-payload-start-task-stream-to-rtsp.json"
Write-Host "mrpcPayloadFile:" $mrpcPayloadFile

mosquitto_pub -h $mqttIpAddress -p $mqttPort -q 1 -V mqttv5 -t $mrpcTopic -D PUBLISH
response-topic $mrpcResponseTopic -f $mrpcPayloadFile
```

You can check the pod log to see the task status:

```
kubectl logs -n $aioConnectorsNamespace -l app.kubernetes.io/component=aio-opc-rtsp-1 --
all-containers --ignore-errors --tail=25
```

Stop the *stream-to-rtsp* task in the asset:

```
$mrpcTopic = "aio-connectors/asset-operations/asset-msft-azure-retail-hls-1/stop-task"
Write-Host "mrpcTopic:" $mrpcTopic

$mrpcResponseTopic = "$mrpcTopic/response"
Write-Host "mrpcResponseTopic:" $mrpcResponseTopic

$mrpcPayloadFile = "mrpc-payload-stop-task-stream-to-rtsp.json"
Write-Host "mrpcPayloadFile:" $mrpcPayloadFile

mosquitto_pub -h $mqttIpAddress -p $mqttPort -q 1 -V mqttv5 -t $mrpcTopic -D PUBLISH
response-topic $mrpcResponseTopic -f $mrpcPayloadFile
```

You can check the pod log to see the task status:

```
kubectl logs -n $aioConnectorsNamespace -l app.kubernetes.io/component=aio-opc-rtsp-1 --
all-containers --ignore-errors --tail=25
```

Get the public address of the media server:

```
$mediaServerJson = kubectl get -n media-server services/media-server-public -o
jsonpath='{.status.loadBalancer.ingress}'
$mediaServerIpAddress = ($mediaServerJson | ConvertFrom-Json).ip
if ($mediaServerIpAddress -eq $null) {
    $mediaServerIpAddress = ($mediaServerJson | ConvertFrom-Json).hostname
}
Write-Host "mediaServerIpAddress:" $mediaServerIpAddress
```

To see the links that you can use to access the stream media, run the following **C#** code block:

```
using System.IO;

#!set --value @pwsh:mediaServerIpAddress --name mediaServerIpAddress

var markdown = @"

## Example commands and links to streams

You can open the stream in different formats and with a variety of clients such as
ffmpeg, vlc, web browsers:
```

```
**RTSP:**
- ffplay rtsp://mediaServerIpAddress:8554/aio-connectors/data/asset-msft-azure-retail-hls-1
- vlc --network-caching=50 rtsp://mediaServerIpAddress:8554/aio-connectors/data/asset-msft-azure-retail-hls-1

**RTMP:**
- ffplay rtmp://mediaServerIpAddress/aio-connectors/data/asset-msft-azure-retail-hls-1

**SRT:**
- ffplay srt://mediaServerIpAddress$mediaServerIpAddress:8890?streamid=read:aio-connectors/data/asset-msft-azure-retail-hls-1
- vlc --network-caching=50 srt://mediaServerIpAddress:8890?streamid=read:aio-connectors/data/asset-msft-azure-retail-hls-1

**HLS:**
- ffplay http://mediaServerIpAddress:8888/aio-connectors/data/asset-msft-azure-retail-hls-1/index.m3u8
- start http://mediaServerIpAddress:8888/aio-connectors/data/asset-msft-azure-retail-hls-1/

";
```

Appendix 1. Datapoint schema grid view

The schema describes the features, here is the grid view of the schema:

$id	https://azure.com/aio/media-connector/datapoint.schema.json
$schema	https://json-schema.org/draft/2020-12/schema
description	Asset.dataPoint.dataPointConfiguration schema
type	object
properties	**properties {}** **taskType {}** type: string
required	**required[1]** taskType
oneOf	**oneOf[5]** **properties** 1 **properties {}** **taskType {}** const: snapshot-to-mqtt **format {}** type: string enum: enum[6] 1 png 2 bmp 3 jpg 4 jpeg 5 tif 6 tiff description: The format of the image. The default is png. **autostart {}** type: boolean description: Whether to start the snapshotting immediately. Default is false. **loop {}**

type	boolean
description	Whether to loop the video clip. The default is false.

fps {}

type	number
minimum	0
description	The number of frames per second to capture. Default is 1. If set to 0, the camera will capture frames in the native frame rate. Example: 30 for a 30 frames per second; 1/60 for one frame per minutes.

width {}

type	integer
minimum	1
description	The width of the image. If not specified, the camera will use the source width.

height {}

type	integer
minimum	1
description	The height of the image. If not specified, the camera will use the source height.

qos {}

type	integer
enum	**enum[3]** 1 0 2 1 3 2
description	The quality of service for the MQTT message. Default is 0.

2 **properties {}**

taskType {}

const	clip-to-mqtt

format {}

type	string
enum	**enum[7]** 1 avi

	2 mp4 3 mkv 4 mts 5 mjpeg 6 mpeg 7 flv
description	The video clip format. The default is avi.

autostart {}

type	boolean
description	Whether to start the clipping immediately. The default is false.

realtime {}

type	boolean
description	Whether to clip in real time. This affects how files get processed, if true the file will be processes as fast possible. The default is false.

loop {}

type	boolean
description	Whether to loop the video clip. The default is false.

fps {}

type	number
minimum	0
description	The number of frames per second to capture. Default is 1. If set to 0, the camera will capture frames in the native frame rate. Example: 30 for a 30 frames per second; 1/60 for one frame per minutes.

width {}

type	integer
minimum	1
description	The width of the image. If not specified, the camera will use the source width.

height {}

type	integer
minimum	1
description	The height of the image. If not specified, the camera will use the source height.

qos {}

type	integer
enum	**enum[3]** 1 0 2 1 3 2
description	The quality of service for the MQTT message. Default is 0.

3 **properties {}**

taskType {}

const	snapshot-to-fs

format {}

type	string
enum	**enum[6]** 1 png 2 bmp 3 jpg 4 jpeg 5 tif 6 tiff
description	The format of the image

autostart {}

type	boolean
description	Whether to start the snapshotting immediately. The default is false.

realtime {}

type	boolean
description	Whether to clip in real time. This affects how files get processed, if true the file will be processes as fast as possible. The default is false.

loop {}

type	boolean
description	Whether to loop the video clip. The default is false.

fps {}

type	number

minimum	0
description	The number of frames per second to capture

width {}

type	integer
minimum	1
description	The width of the image

height {}

type	integer
minimum	1
description	The height of the image

path {}

type	string
description	The path to save the image, by default set by the broker

4 **properties {}**

taskType {}

const	clip-to-fs

format {}

type	string
enum	**enum[7]** 1 avi 2 mp4 3 mkv 4 mts 5 mjpeg 6 mpeg 7 flv
description	The video clip format. The default is avi.

autostart {}

type	boolean
description	Whether to start the clipping immediately. The default is false.

realtime {}

type	boolean

description	Whether to clip in real time. This affects how files get processed, if true the file will be processes as fast as possible. The default is false.

loop {}

type	boolean
description	Whether to loop the video clip. The default is false.

fps {}

type	number
minimum	0
description	The number of frames per second to capture

width {}

type	integer
minimum	1
description	The width of the video clip

height {}

type	integer
minimum	1
description	The height of the video clip

duration {}

type	integer
minimum	1
description	The duration of each video clip segment

path {}

type	string
description	The path to save the image, by default set by the broker

5 **properties {}**

taskType {}

const	stream-to-rtsp

autostart {}

type	boolean
description	Whether to start the clipping immediately. The default is false.

realtime {}

type	boolean
description	Whether to clip in real time. This affects how files get processed, if true the file will be processes as fast possible. The default is false.

loop {}

type	boolean
description	Whether to loop the video clip. The default is false.

fps {}

type	number
minimum	0
description	The number of frames per second to capture

width {}

type	integer
minimum	1
description	The width of the video clip

height {}

type	integer
minimum	1
description	The height of the video clip

media_server_address {}

type	string
description	The path to save the image, by default set by the broker

media_server_port {}

type	integer
minimum	1
description	The media server port

Appendix 2. media-connector-vars.ps1 sample code

media-connector-vars.ps1

```
$aioConnectorsNamespace = "aio-connectors"
Write-Host "Using namespace: $aioConnectorsNamespace"
Write-Host "`n"

$aioRuntimeVersion = "0.6.0-preview.4"
Write-Host "Using aio-runtime version: $aioRuntimeVersion"
Write-Host "`n"

$mqttAddress = 'mqtt://mosquitto.' + $aioConnectorsNamespace + ':1883'
Write-Host "Using MQTT address: $mqttAddress"
Write-Host "`n"

Write-Host "Media Connector Configuration:"
$mediaConnectorImageRegistryName = "aioconnectorsdev"
Write-Host "Media Connector Image Registry Name: $mediaConnectorImageRegistryName"
$mediaConnectorImageRegistryServer = "$mediaConnectorImageRegistryName.azurecr.io"
Write-Host "Media Connector Image Registry Server: $mediaConnectorImageRegistryServer"
$mediaConnectorImageRepository = "artifact/d065c211-cbf1-43f4-85d4-
e52d2844e743/dev/media-broker"
Write-Host "Media Connector Image Repository: $mediaConnectorImageRepository"
$mediaConnectorImageTag = "0.1.0-alpha.dev.2024-06-12"
Write-Host "Media Connector Image Tag: $mediaConnectorImageTag"
$mediaConnectorConfigurationSchema = "https://aiobrokers.blob.core.windows.net/aio-
media-connector/1.0.0.json"
Write-Host "Media Connector Configuration Schema: $mediaConnectorConfigurationSchema"
Write-Host "`n"

Write-Host "Monitoring Configuration:"
$monitoringNamespace = "aio-brokers-monitoring"
Write-Host "Using monitoring namespace: $monitoringNamespace"
$otelCollectorEndpoint = 'http://otel-collector.' + $monitoringNamespace +
'.svc.cluster.local:4317'
Write-Host "Using OpenTelemetry collector endpoint: $otelCollectorEndpoint"
$connectorDeploymentName = "aio-brokers-sample-none"
Write-Host "Using connector deployment name: $connectorDeploymentName"
Write-Host "`n"
```

Appendix 3. aep-msft-azure-retail-hls-1.yaml sample code

aep-msft-azure-retail-hls-1.yaml

```yaml
apiVersion: deviceregistry.microsoft.com/v1beta1
kind: AssetEndpointProfile
metadata:
  name: aep-msft-azure-retail-hls-1
spec:
  additionalConfiguration: |-
    {
      "$schema": "https://aiobrokers.blob.core.windows.net/aio-media-
connector/1.0.0.json"
    }
  targetAddress: http://s7d2.scene7.com/is/content/microsoftcorp/CLO24-Azure-Retail-16x9
  transportAuthentication:
    ownCertificates: []
  userAuthentication:
    mode: Anonymous
    usernamePasswordCredentials:
      usernameReference: ""
      passwordReference: ""
  uuid: 1ce383eb-2a06-4be5-afc1-b79a14dccedb
```

Appendix 4. asset-msft-azure-retail-hls-1.yaml sample code

asset-msft-azure-retail-hls-1.yaml

```yaml
apiVersion: deviceregistry.microsoft.com/v1beta1
kind: Asset
metadata:
  name: asset-msft-azure-retail-hls-1
spec:
  displayName: Microsoft Azure Corporate Video
  description: Microsoft Azure Corporate Video
  assetEndpointProfileUri: aep-msft-azure-retail-hls-1
  dataPoints:
    - dataSource: snapshot-to-mqtt
      capabilityId: snapshot-to-mqtt
      name: snapshot-to-mqtt
      dataPointConfiguration: |-
        {
          "autostart": "false",
          "fps": "1",
          "format": "jpeg",
          "realtime": "true",
          "loop": "true"
        }
    - dataSource: snapshot-to-fs
      capabilityId: snapshot-to-fs
      name: snapshot-to-fs
      dataPointConfiguration: |-
        {
          "autostart": "false",
          "fps": "1",
          "format": "jpeg",
          "realtime": "true",
          "loop": "true"
        }
    - dataSource: clip-to-fs
      capabilityId: clip-to-fs
      name: clip-to-fs
      dataPointConfiguration: |-
        {
          "autostart": "false",
          "format": "mkv",
```

```
      "duration": "60",
      "realtime": "true",
      "loop": "true"
    }
- dataSource: stream-to-rtsp
  capabilityId: stream-to-rtsp
  name: stream-to-rtsp
  dataPointConfiguration: |-
    {
      "autostart": "false",
      "realtime": "true",
      "loop": "true"
    }
```

Appendix 5. JSON examples collection

mrpc-payload-start-task-snapshot-to-fs.json

```
{
    "datapoint": "snapshot-to-fs"
}
```

mrpc-payload-start-task-snapshot-to-mqtt.json

```
{
    "datapoint": "snapshot-to-mqtt"
}
```

mrpc-payload-start-task-stream-to-rtsp.json

```
{
    "datapoint": "stream-to-rtsp"
}
```

mrpc-payload-stop-task-snapshot-to-fs.json

```
{
    "datapoint": "snapshot-to-fs"
}
```

mrpc-payload-stop-task-snapshot-to-mqtt.json

```
{
    "datapoint": "snapshot-to-mqtt"
}
```

mrpc-payload-stop-task-stream-to-rtsp.json

```
{
    "datapoint": "stream-to-rtsp"
}
```