**aws**

# Media Insights on AWS

# Media Insights on AWS: Implementation Guide

# Table of Contents

# Development framework for building applications that process multimedia

Publication date: *April 2021* ([last update](): *November 2024*)

Media Insights on AWS is a development framework for building applications that process videos, images, audio, and text with machine learning services on Amazon Web Services (AWS). Media Insights on AWS takes care of workflow orchestration and data persistence so that you can focus on workflow development. By addressing the concerns of running workflows, Media Insights on AWS empowers you to build applications faster with the benefit of inheriting a pre-built and robust backend.

For additional details and sample use cases, refer to [How to rapidly prototype multimedia applications with Media Insights on AWS]() on the AWS Media Blog.

This solution does not provide a graphical user interface (GUI); however, the Media Insights on AWS [GitHub repository]() includes a Media Insights on AWS reference application that features a GUI. Consider using this for your own implementation.

This implementation guide provides an overview of the Media Insights on AWS solution, its reference architecture and components, considerations for planning the deployment, and configuration steps for deploying the Media Insights on AWS solution to the AWS Cloud.

The intended audience for using this solution's features and capabilities in their environment includes solution architects, business decision makers, DevOps engineers, data scientists, and cloud professionals.

Use this navigation table to quickly find answers to these questions:

| If you want to . . . | Read . . . |
|---|---|
| Know the cost for running this solution.<br><br>The estimated cost for running this solution in the US-East (N.Virginia) Region is USD $24.00 per month. | [Cost]() |

| If you want to . . . | Read . . . |
|---|---|
| Understand the security considerations for this solution. | [Security](#) |
| Know how to plan for quotas for this solution. | [Quotas](#) |
| Know which AWS Regions are supported for this solution. | [Supported AWS Regions](#) |
| View or download the AWS CloudForm ation template included in this solution to automatically deploy the infrastructure resources (the "stack") for this solution. | [AWS CloudFormation template](#) |
| Access the source code and optionally use the AWS Cloud Development Kit (AWS CDK) to deploy the solution. | [GitHub repository](#) |

# Features and benefits

The solution provides the following features:

**Reduce development time**

This solution manages workflow orchestration and data persistence so that you can focus on applications that extract value from media or automate manual workflows.

**Customization**

You can extend and customize the solution to fit new use cases. Operators are generated state machines that are pre-built, but you can also extend them to handle specific use cases.

**Modular framework**

Components are described by clean interfaces. Operators are small single-purpose components that transform or extract metadata from media. You can define custom operators or use any of the included pre-built operators.

**Integration with AWS Service Catalog AppRegistry and AWS Systems Manager Application Manager**

Media Insights on AWS includes a Service Catalog AppRegistry resource to register the solution's CloudFormation template and its underlying resources as an application in both Service Catalog AppRegistry and AWS Systems Manager Application Manager. With this integration, you can centrally manage the solution's resources.

# Use cases

### Transforming video content with redaction

Use Amazon Rekognition to identify faces, potentially unsafe, inappropriate or unwanted content; extract and blur video frames; and stitch the blurred frames into the original video.

### Deriving video features for ad placement

Find scenes in video where it might be appropriate to insert ads. By using OpenCV to capture pixel data and Amazon Rekognition to extract video metadata, Media Insights on AWS can generate an ad placement Video Multiple Ad Playlist (VMAP) file that can be imported into advertising systems.

### Evaluating off-the-shelf artificial intelligence (AI) with your own content

Use pre-trained AI services without prior machine learning experience. You can use Content Localization on AWS, or build your own application, on top of the Media Insights on AWS framework to test-drive AWS AI services with your own content.

### Using speech-to-text services to generate subtitles

Use Amazon Transcribe to automatically generate Web Video Text Tracks (WebVTT) and SubRip text (SRT) format subtitles. You can edit the subtitles interactively with Content Localization on AWS, or your own web application, and save them to invoke reprocessing of downstream operators in the workflow.

### Indexing videos based on visual and audio content

Turn media archives into searchable assets for content discovery and monetization.

# Concepts and definitions

This section describes key concepts and defines terminology specific to this solution:

**application**

A logical group of AWS resources that you want to operate as a unit.

**workflow application programming interface (API)**

A REST interface to create, update, delete, and run workflows and operators, and monitor workflows.

**control plane**

Includes the workflow API and state machines for workflows. Workflow state machines are composed of operators from the Media Insights on AWS operator library. When operators within the state machine are run, they interact with the Media Insights on AWS data plane to store and retrieve derived asset and metadata generated from the workflow.

**operators**

Operators are generated state machines that call AWS Lambda functions to perform media analysis or media transformation tasks.

**workflows**

Generated state machines that run a number of operators in sequence.

**data plane**

Stores media assets and their associated metadata that are generated by workflows. Implement a consumer of the Amazon Kinesis Data Streams in the data plane to extract, transform, and load (ETL) data from the master Media Insights on AWS data store to downstream databases that support the data access patterns required by end-user applications.

**data plane API**

A REST interface to create, update, delete, and retrieve media assets and their associated metadata.

**data plane pipeline**

Stores metadata for an asset that can be retrieved as a single block or pages of data using an object's AssetId and Metadata type. Writing data to the pipeline initiates a copy of the data to

be stored in Kinesis Data Streams. End-user applications can connect to this data stream as an interface to use data stored in the Media Insights on AWS data plane.

**data pipeline consumer**

Changes to the data plane Amazon DynamoDB table are reflected in a Kinesis data stream.

For each record in that data stream, data pipeline consumers perform the necessary ETL tasks needed to replicate data, such as media metadata, to the data stores used by external applications. These ETL tasks are use case dependent and therefore must be user-defined.

For a general reference of AWS terms, see the AWS Glossary.

# Architecture overview

This section provides a reference implementation architecture diagram for the components deployed with this solution.

# Architecture diagram

Deploying this solution with the default parameters deploys the following components in your AWS account.
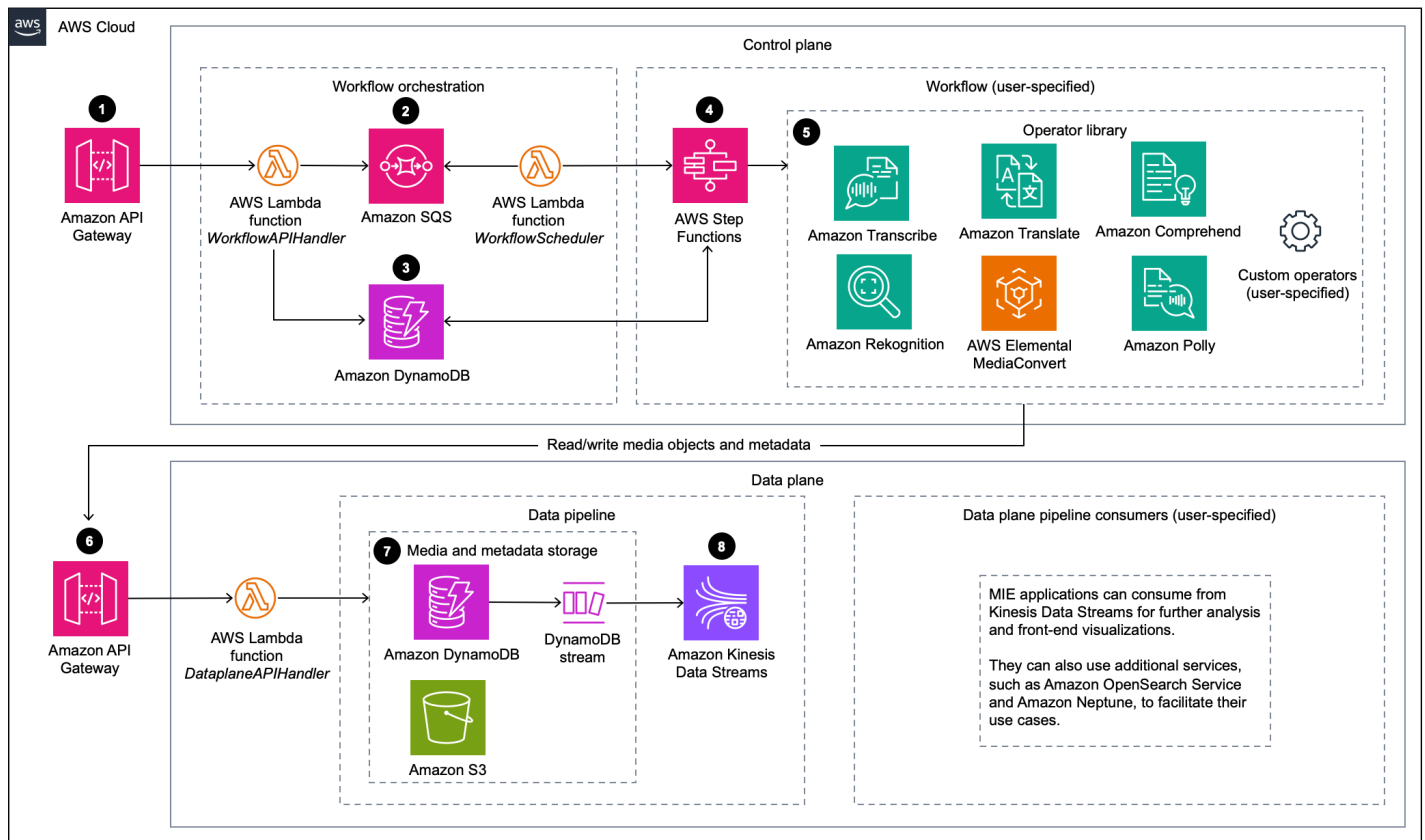


*Media Insights on AWS architecture*

> ⓘ **Note**
>
> AWS CloudFormation resources are created from AWS Cloud Development Kit (AWS CDK) constructs.

The high-level process flow for the solution components deployed with the AWS CloudFormation template is as follows:

1. An Amazon API Gateway resource for the control plane REST API. This resource is the entry point where requests to create, read, update, delete (CRUD) or run workflows begin.

2. AWS Lambda and Amazon Simple Queue Service (Amazon SQS) resources to support workflow orchestration and translating user-defined workflows into AWS Step Functions. A Lambda function updates workflow-related tables in Amazon DynamoDB, at which point CRUD workflow requests finish. Requests to run workflows begin when another Lambda function saves the request to an Amazon SQS queue. The queue is later read and run by the WorkflowScheduler Lambda function that controls how many workflows can run at the same time.

3. Amazon DynamoDB tables to store workflow-related data, such as state machine definitions for operators, workflow configurations, and workflow run status.

4. A Step Functions resource that is created when a user defines a new workflow using the workflow API. When the WorkflowScheduler Lambda function starts a workflow, it starts the Step Functions resource, which then invokes a series of Lambda functions that call external services and download results from those services. When all of the Lambda functions in a workflow have fully run, then a Lambda function is called to update the workflow status in DynamoDB.

5. Lambda functions for using the following commonly used services in Media Insights on AWS workflows: Amazon Rekognition, Amazon Comprehend, Amazon Translate, Amazon Transcribe, Amazon Polly, and AWS Elemental MediaConvert. Operators consist of Lambda functions that call external services and download results from those services. They are invoked by a state machine in Step Functions, as prescribed by the workflow definition. These Lambda functions save results to long-term storage via the data plane REST API.

6. An Amazon API Gateway resource for the data plane REST API. Operators save results to long-term storage by calling this API.

7. Amazon Simple Storage Service (Amazon S3), DynamoDB, and DynamoDB Streams for media and metadata data storage. The Lambda function behind the data plane API directly accesses Amazon S3 and DynamoDB to perform incoming CRUD requests. That Lambda function saves files, such as binary media files or JSON metadata files, in Amazon S3. A pointer to those files is saved in a DynamoDB table. Finally, a time-ordered sequence of modifications to that table are saved in a DynamoDB Stream and an Amazon Kinesis Data Streams resource.

8. Amazon Kinesis Data Streams for interfacing with external applications. A Kinesis data stream provides an interface for external applications to access data stored in the Media Insights on

AWS data plane. This interface is appropriate for feeding downstream data stores, such as the Amazon OpenSearch Service or Amazon Neptune, that support specialized data access patterns required by end-user applications. To feed a downstream data store, you must implement a consumer (for example, a Lambda function) that consumes records from the data stream and performs the necessary ETL tasks needed for the external application.

> ⓘ **Note**
>
> The ETL tasks that feed downstream data stores are use case dependent and therefore must be user-defined. The Media Insights on AWS Developer guide includes detailed instructions for implementing ETL functions in Media Insights on AWS.

# AWS Well-Architected design considerations

We designed this solution with best practices from the AWS Well-Architected Framework, which helps customers design and operate reliable, secure, efficient, and cost-effective workloads in the cloud.

This section describes how the design principles and best practices of the Well-Architected Framework were applied when building this solution.

## Operational excellence

This section describes how we architected this solution using the principles and best practices of the operational excellence pillar.

The Media Insights on AWS solution pushes metrics to Amazon CloudWatch at various stages to provide observability into the infrastructure, Lambda functions, AI services, Amazon S3 buckets, and the rest of the solution components.

## Security

This section describes how we architected this solution using the principles and best practices of the security pillar.

AWS highly recommends that customers encrypt sensitive data in transit and at rest. Media Insights on AWS automatically encrypts media files and metadata at rest with Amazon S3 server-side encryption (SSE).

The Media Insights on AWS solution's Amazon SNS topics and Amazon DynamoDB tables are also encrypted at rest using SSE.

Media Insights on AWS uses [AWS Identity and Access Management](#) (AWS IAM) to authorize REST API requests. Refer to the documentation for your chosen HTTP client to learn how to use IAM in your application.

To adhere to security best practices, the solution's stack creates a dedicated [AWS Key Management Service](#) (AWS KMS) customer-managed key in your account. Therefore, to access data derived by Media Insights on AWS outside of using Media Insights on AWS interfaces, such as APIs and data pipelines, you must use (and have access to) that encryption key to decrypt the data. If that key gets deleted, that data will be irrecoverable. Rotation of customer managed keys is the responsibility of the customer. For more information, refer to [AWS KMS concepts](#) in the *AWS KMS Developer Guide*.

## Reliability

This section describes how we architected this solution using the principles and best practices of the [reliability pillar](#).

Media Insights on AWS uses AWS serverless services wherever possible (for example, Lambda, API Gateway, Amazon S3, and DynamoDB) to ensure high availability and quick recovery from service failure.

## Performance efficiency

This section describes how we architected this solution using the principles and best practices of the [performance efficiency pillar](#).

Media Insights on AWS, as mentioned earlier, uses serverless architecture throughout the solution.

You can launch Media Insights on AWS in any Region that supports the AWS services used in the solution such as: AWS Lambda, Amazon API Gateway, Amazon S3, Amazon Rekognition, Amazon Translate, Amazon Transcribe, Amazon Comprehend, Amazon Polly, and AWS Elemental MediaConvert.

This solution is automatically tested and reviewed by solutions architects and subject matter experts for areas to experiment and improve.

# Cost optimization

This section describes how we architected this solution using the principles and best practices of the cost optimization pillar.

Media Insights on AWS uses serverless architecture; therefore customers only get charged for what they use.

The solution is a modular framework that allows users to configure and tailor their own media workflows, and using only the AWS services that they need.

# Sustainability

This section describes how we architected this solution using the principles and best practices of the sustainability pillar.

Media Insights on AWS uses managed and serverless services to minimize the environmental impact of the backend services. Maximizing the usage of the AWS AI services is a critical component for sustainability provided by the solution. The serverless design of Media Insights on AWS (using Lambda, API Gateway, Amazon S3, and DynamoDB) are aimed at reducing carbon footprint compared to the footprint of continually operating on-premises servers.

# Architecture details

This section describes the components and AWS services that make up this solution and the architecture details of how these components work together.

## Control plane

This includes the workflow API and state machines for workflows. Workflow state machines are composed of operators from the Media Insights on AWS operator library. When operators within the state machine are run, they interact with the Media Insights on AWS data plane to store and retrieve derived asset and metadata generated from the workflow.

Use the control plane to CRUD custom operators and workflows, and to execute those workflows.

## Data plane

This stores the media assets and metadata generated by workflows. Implement a consumer of the Kinesis data stream in the data plane to ETL data from the master Media Insights on AWS data store to downstream databases that support the data access patterns required by end-user applications.

## Data plane pipeline

This pipeline stores metadata for an asset that can be retrieved using an object's `AssetId` and `Metadata` type. Writing data to the pipeline initiates a copy of the data to be stored in Kinesis Data Streams. End-user applications can connect to this data stream as an interface to use data stored in the Media Insights on AWS data plane.

## Data pipeline consumers

Changes to the data plane DynamoDB table are reflected in a Kinesis data stream. For each record in that data stream, data pipeline consumers perform the necessary ETL tasks needed to replicate data, such as media metadata, to the data stores used by external applications. These ETL tasks are use-case dependent and therefore must be user-defined. The [Media Insights on AWS Developer guide](#) includes detailed instructions for implementing data plane pipeline consumers.

# AWS services in this solution

| AWS service | Description |
| --- | --- |
| [Amazon API Gateway](#) | **Core.** Entrypoint to interact with the control and data plane APIs where requests to create, read, update, delete, or run workflows begin, or data retrieval begin. |
| [Amazon Comprehend](#) | **Core.** Can be integrated into workflows to find key phrases in text and references to real-world objects, dates, and quantities in text. |
| [Amazon DynamoDB](#) | **Core.** Stores workflow-related data, such as state machine definitions for operators, workflow configurations, and workflow run status. |
| [Amazon Elemental MediaConvert](#) | **Core.** Can be integrated into workflows to transcode input video into MPEG4 format and generate thumbnails. |
| [AWS Identity and Access Management (IAM)](#) | **Core.** Grants the solution's AWS Lambda function access to create Regional resources. |
| [Amazon Kinesis Data Streams](#) | **Core.** Utilized to stream data changes reflected in DynamoDB which consumers can extract, transform, and load data from the data store to downstream services. |
| [AWS Lambda](#) | **Core.** Supports workflow orchestration, operators executions, and store workflow results. |
| [Amazon Polly](#) | **Core.** Can be integrated into workflows to turn input text into speech. |
| [Amazon Rekognition](#) | **Core.** Can be integrated into workflows for celebrity recognition, content moderation, |

| AWS service | Description |
| --- | --- |
| | face detection, face search, label detection, person tracking, shot, text, and technical cue detection. |
| Amazon Simple Notification Service | **Core.** Used for supporting Workflow execution rate limits. |
| Amazon Simple Queue Service | **Core.** Used for supporting Workflow execution rate limits. |
| Amazon Simple Storage Service | **Core.** Resource used for storing input user media and output transformed media by the workflow. |
| AWS Step Functions | **Core.** Resource that is created when a user defines a new workflow using the workflow API. |
| AWS Systems Manager | **Core.** Provides application-level resource monitoring and visualization of resource operations and cost data. |
| Amazon Transcribe | **Core.** Can be integrated into workflows to create SRT or VTT caption files from video transcripts. It can also convert input audio to text. |
| Amazon Translate | **Core.** Can be integrated into workflows to translate input text. |
| AWS X-Ray | **Core.** Provides debugging tools for the Media Insights on AWS application. |

# Plan your deployment

This section describes the cost, security, Region, and quota considerations for planning your deployment.

## Cost

You are responsible for the cost of the AWS services used while running this solution. As of this revision, the cost for running this solution with the default settings in the US East (N. Virginia) Region is approximately **$24.00 per month without free tiers, or $13.00 per month with free tiers for 100 workflow runs**. Add approximately $2.40 for each additional 100 workflow runs. Most Media Insights on AWS use cases are covered by the free tier for all AWS services except Amazon Kinesis and AWS Lambda.

We recommend creating a budget through AWS Cost Explorer to help manage costs. Prices are subject to change. For full details, refer to the pricing webpage for each AWS service used in this solution.

### Sample cost table

The following table provides a sample cost breakdown for deploying this solution with the default parameters in the US East (N. Virginia) Region for one month.

| AWS service | Dimensions | Cost [USD] / month |
|---|---|---|
| Amazon API Gateway | 1,000,000 workflows | $3.50 |
| Amazon DynamoDB | 1,000,000 workflows | $0.25 |
| Amazon Lambda | 100 workflows | $4.75 |
| Amazon Kinesis | 100 workflows | $12.56 |
| Amazon SQS | 1,000,000 workflows | $0.40 |
| Amazon SNS | 1,000,000 workflows | $0.00 |
| Amazon S3 | 100 workflows | $2.30 |

| AWS service | Dimensions | Cost [USD] / month |
|-------------|------------|--------------------|
| AWS X-Ray | 100 workflows | $0.0005 |
| | **Total** | **~$23.76** |

# Security

When you build systems on AWS infrastructure, security responsibilities are shared between you and AWS. This [shared responsibility model](#) reduces your operational burden because AWS operates, manages, and controls the components including the host operating system, virtualization layer, and physical security of the facilities in which the services operate. For more information about AWS security, visit [AWS Cloud Security](#).

## IAM roles

IAM roles allow customers to assign granular access policies and permissions to services and users on the AWS Cloud. This solution creates IAM roles that grant the solution's AWS Lambda functions access to create Regional resources.

# Supported AWS Regions

This solution uses the AWS Clean Rooms service, which is not currently available in all AWS Regions. You must launch this solution in an AWS Region where AWS Clean Rooms is available. For the most current availability of AWS services by Region, refer to the [AWS Regional Services List](#).

Data Connectors for AWS Clean Rooms is supported in the following AWS Regions:

| Region name | |
|-------------|--|
| US East (Ohio) | Europe (London) |
| US East (N. Virginia) | Asia Pacific (Mumbai) |
| US West (Oregon) | Asia Pacific (Seoul) |
| Canada (Central) | Asia Pacific (Singapore) |

| Region name | |
|---|---|
| Europe (Frankfurt) | Asia Pacific (Sydney) |
| Europe (Ireland) | Asia Pacific (Tokyo) |

# Quotas

Service quotas, also referred to as limits, are the maximum number of service resources or operations for your AWS account.

## Quotas for AWS services in this solution

Make sure you have sufficient quota for each of the services implemented in this solution. For more information, refer to AWS service quotas.

Click one of the following links to go to the page for that service. To view the service quotas for all AWS services in the documentation without switching pages, view the information in the Service endpoints and quotas page in the PDF instead.

## AWS CloudFormation quotas

Your AWS account has AWS CloudFormation quotas that you should be aware of when launching the stack in this solution. By understanding these quotas, you can avoid limitation errors that would prevent you from deploying this solution successfully. For more information, refer to AWS CloudFormation quotas in the *AWS CloudFormation Users Guide*.

# Deploy the solution

This solution uses [AWS CloudFormation templates and stacks](#) to automate its deployment. The CloudFormation template describes the AWS resources included in this solution and their properties. The CloudFormation stack provisions the resources that are described in the template.

> ⚠️ **Important**
>
> This solution includes an option to send anonymized operational metrics to AWS. We use this data to better understand how customers use this solution and related services and products. AWS owns the data gathered though this survey. Data collection is subject to the [AWS Privacy Notice](#).
> To opt out of this feature, download the template, modify the AWS CloudFormation mapping section, and then use the AWS CloudFormation console to upload your updated template and deploy the solution. For more information, see the [Anonymized data collection](#) section of this guide.

# AWS CloudFormation template

You can download the CloudFormation template for this solution before deploying it.

[View template]

**media-insights-on-aws-stack.template** - Use this template to launch the solution and all associated components. The default configuration deploys Amazon API Gateway, AWS Lambda, Amazon SNS, Amazon SQS, Amazon DynamoDB, AWS Step Functions, Amazon S3, Amazon Kinesis Data Streams, and AWS IAM, but you can customize the template to meet your specific needs.

> ⓘ **Note**
>
> AWS CloudFormation resources are created from AWS CDK constructs.

This AWS CloudFormation template deploys Media Insights on AWS in the AWS Cloud.

> **ⓘ Note**
>
> If you have previously deployed this solution, see *Update the solution* for update
> instructions.

# Deployment process overview

Before you launch the solution, review the [cost](#), [architecture](#), [network security](#), and other
considerations discussed earlier in this guide.

Follow the step-by-step instructions in this section to configure and deploy the solution into your
account.

**Time to deploy:** Approximately 10 minutes

> **ⓘ Note**
>
> If you have previously deployed this solution, see [Update the solution](#) for update
> instructions.

# Launch the stack

Follow the step-by-step instructions in this section to configure and deploy the solution into your
account.

**Time to deploy:** Approximately 10 minutes

1. Sign in to the [AWS Management Console](#) and select the button to launch the `media-insights-on-aws-stack.template` AWS CloudFormation template.

   [**Launch solution**]

2. The template launches in the US East (N. Virginia) Region by default. To launch the solution in a
   different AWS Region, use the Region selector in the console navigation bar.

> **ⓘ Note**
>
> This solution uses Amazon Comprehend, Amazon Translate, Amazon Transcribe, Amazon Polly, AWS Elemental MediaConvert, and Amazon Rekognition, which are not currently available in all AWS Regions. You must launch this solution in an AWS Region where these services are available. For the most current availability by Region, see the AWS Regional Services List.

3. On the **Create stack** page, verify that the correct template URL is in the **Amazon S3 URL** text box and choose **Next**.

4. On the **Specify stack details** page, assign a name to your solution stack. For information about naming character limitations, see IAM and STS Limits in the *AWS Identity and Access Management User Guide*.

5. Under **Parameters**, review the parameters for this solution template and modify them as necessary. This solution uses the following default values.

| Parameter | Default | Description |
|---|---|---|
| **Max Concurrent Workflows** | 5 | Identifies the maximum number of workflows to run concurrently. When the maximum is reached, additional workflows are added to a wait queue. The recommended range is 2 to 5. |
| **Deploy Analytics Pipeline** | true | Determines whether to deploy a metadata streaming pipeline that can be consumed by downstream analytics platforms. By default, this capability is activated when the solution |

| Parameter | Default | Description |
|---|---|---|
|  |  | is deployed. Set to false to deactivate this capability. |
| **Deploy Test Resources** | `false` | Determines whether to deploy test resources that contain Lambda functions required for integration and end-to-end testing. By default, this capability is deactivated when the solution is deployed. Set to true to activate this capability. |
| **Turn on X-Ray Trace** | `false` | Determines whether to activate Active X-Ray tracing on all entry points to the stack. By default, this capability is deactivated when the solution is deployed. Set to true to activate this capability. |
| **External Bucket ARN** | *<Optional input>* | Specify the Amazon S3 Amazon Resources Name (ARN) for media files that Media Insights on AWS needs permission to read as inputs to workflows. Leave blank if the Media Insights on AWS data plane bucket is the only source for media files. |

6. Choose **Next**.

7. On the **Configure stack options** page, choose **Next**.

8. On the **Review and create** page, review and confirm the settings. Select the box acknowledging that the template will create AWS IAM resources.

9. Choose **Submit** to deploy the stack.

You can view the status of the stack in the AWS CloudFormation console in the **Status** column. You should receive a CREATE_COMPLETE status in approximately 10 minutes.

# Monitor the solution with Service Catalog AppRegistry

This solution includes a Service Catalog AppRegistry resource to register the CloudFormation template and underlying resources as an application in both Service Catalog AppRegistry and AWS Systems Manager Application Manager.

AWS Systems Manager Application Manager gives you an application-level view into this solution and its resources so that you can:

- Monitor its resources, costs for the deployed resources across stacks and AWS accounts, and logs associated with this solution from a central location.

- View operations data for the resources of this solution (such as deployment status, CloudWatch alarms, resource configurations, and operational issues) in the context of an application.

  The following figure depicts an example of the application view for the solution stack in Application Manager.



*Solution stack in Application Manager*

# Activate CloudWatch Application Insights

1. Sign in to the Systems Manager console.

2. In the navigation pane, choose **Application Manager**.

3. In **Applications**, search for the application name for this solution and select it.

   The application name will have App Registry in the **Application Source** column, and will have a combination of the solution name, Region, account ID, or stack name.

4. In the **Components** tree, choose the application stack you want to activate.

5. In the **Monitoring** tab, in **Application Insights**, select **Auto-configure Application Insights**.



Monitoring for your applications is now activated and the following status box appears:

# Confirm cost tags associated with the solution

After you activate cost allocation tags associated with the solution, you must confirm the cost allocation tags to see the costs for this solution. To confirm cost allocation tags:

1. Sign in to the [Systems Manager console](#).

2. In the navigation pane, choose **Application Manager**.

3. In **Applications**, choose the application name for this solution and select it.

4. In the **Overview** tab, in **Cost**, select **Add user tag**.
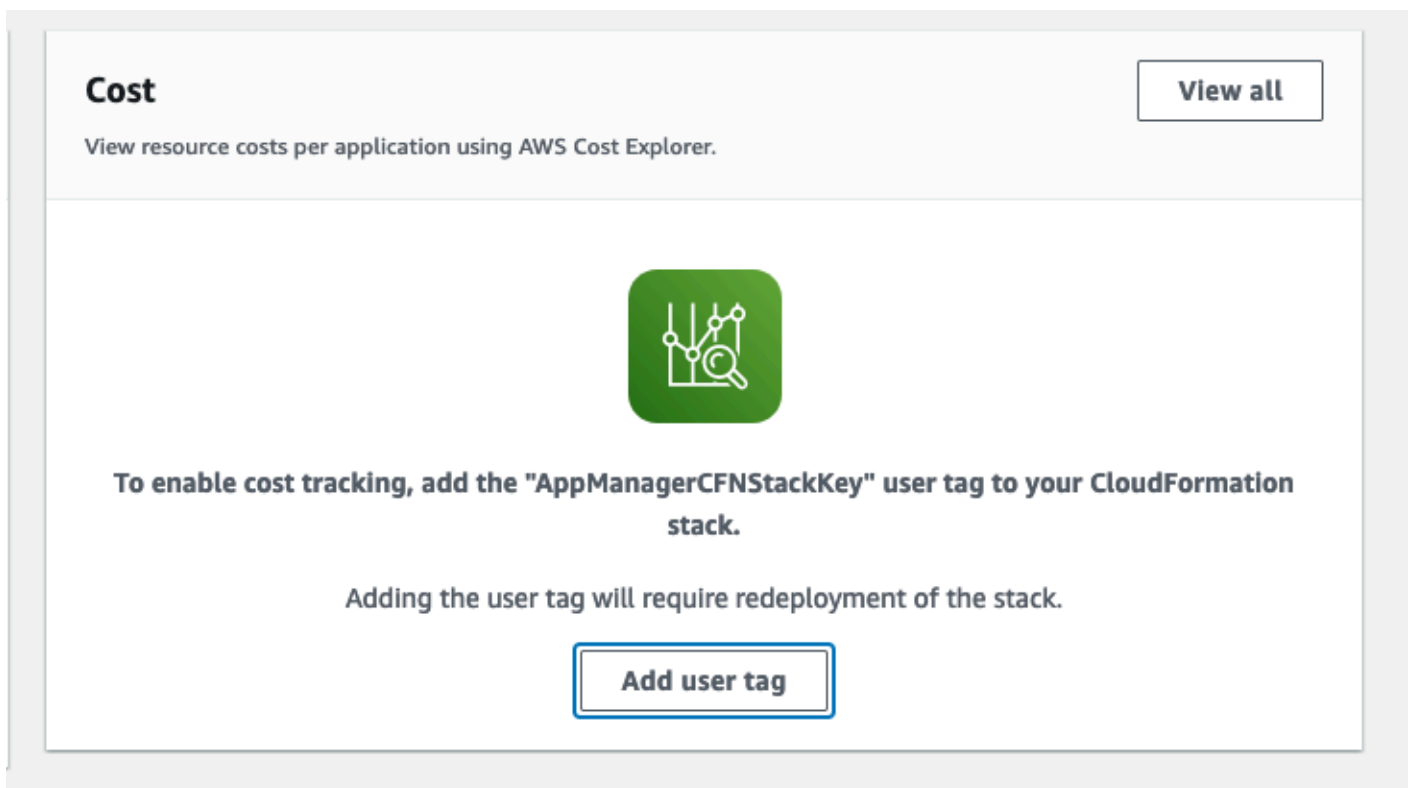


5. On the **Add user tag** page, enter `confirm`, then select **Add user tag**.

The activation process can take up to 24 hours to complete and the tag data to appear.

# Activate cost allocation tags associated with the solution

After you confirm the cost tags associated with this solution, you must activate the cost allocation tags to see the costs for this solution. The cost allocation tags can only be activated from the management account for the organization.

To activate cost allocation tags:

1. Sign in to the [AWS Billing and Cost Management and Cost Management console](#).

2. In the navigation pane, select **Cost Allocation Tags**.

3. On the **Cost allocation tags** page, filter for the `AppManagerCFNStackKey` tag, then select the tag from the results shown.

4. Choose **Activate**.

# AWS Cost Explorer

You can see the overview of the costs associated with the application and application components within the Application Manager console through integration with AWS Cost Explorer. Cost Explorer helps you manage costs by providing a view of your AWS resource costs and usage over time.

1. Sign in to the [AWS Cost Management console](#).

2. In the navigation menu, select **Cost Explorer** to view the solution's costs and usage over time.

# Update the solution

If you have previously deployed the solution, follow this procedure to update the Media Insights on AWS CloudFormation stack to get the latest version of the solution's framework.

1. Sign in to the [AWS CloudFormation console](), select your existing Media Insights on AWS CloudFormation stack, and select **Update**.

2. Select **Replace current template**.

3. Under **Specify template:**

   a. Select **Amazon S3 URL**.

   b. Copy the link of the [latest template]().

   c. Paste the link in the **Amazon S3 URL** box.

   d. Verify that the correct template URL shows in the **Amazon S3 URL** text box, and choose **Next**. Choose **Next** again.

4. Under **Parameters**, review the parameters for the template and modify them as necessary. For details about the parameters, refer to [Launch the Stack]().

5. Choose **Next**.

6. On the **Configure stack options** page, choose **Next**.

7. On the **Review** page, review and confirm the settings. Select the box acknowledging that the template will create IAM resources.

8. Choose **View change set** and verify the changes.

9. Choose **Update stack** to deploy the stack.

You can view the status of the stack in the AWS CloudFormation console in the **Status** column. You should receive a UPDATE_COMPLETE status in approximately 10 minutes.

# Troubleshooting

This section includes information about how to activate AWS X-Ray request tracing, workflow error handling, and related troubleshooting steps.

If these instructions don't address your issue, Contact AWS Support provides instructions for opening an AWS Support case for this solution.

# How to activate AWS X-Ray request tracing for Media Insights on AWS

AWS X-Ray traces requests through the AWS platform. It is especially useful for performance debugging. It also helps with other types of debugging by making it easy to follow what happened with a request end to end across AWS services, even when the invoked request runs across multiple AWS accounts.

The AWS X-Ray service has a perpetual free tier. When free tier limits are exceeded, X-Ray tracing incurs charges as outlined by the X-Ray pricing page.

## Activate tracing from Lambda entry points

By default, tracing for Media Insights on AWS is deactivated. You can activate AWS X-Ray tracing for Media Insights on AWS requests by updating the Media Insights on AWS stack with the **EnableXrayTrace** CloudFormation parameter to true. When tracing is activated, all supported services that are invoked for the request will be traced starting from Media Insights on AWS Lambda entry points. These entry point Lambda functions are as follows:

- WorkflowAPIHandler
- WorkflowCustomResource
- WorkflowScheduler
- DataplaneAPIHandler

## Activate tracing from API Gateway entry points

Additionally, you can activate tracing for API Gateway requests in the AWS Management Console by selecting **Enable tracing** for the deployed API Gateway stages for both the Workflow API and

the Data plane API. For details, refer to [Amazon API Gateway active tracing support for AWS X-Ray](#) in the *AWS X-Ray Developer Guide*.

## Developing custom tracing in Media Insights on AWS Lambda functions

Media Insights on AWS Lambda functions import the [AWS X-Ray SDK for Python](#) packages and patch any supported libraries at runtime. Media Insights on AWS Lambda functions are ready for future instrumentation by developers using the X-Ray Python packages.

The Media Insights on AWS [Lambda layer](#) contains all the packages dependencies needed to support X-Ray, so they are available to any new Lambda functions that use the layer.

# Media Insights on AWS workflow error handling

When you create Media Insights on AWS workflows, Media Insights on AWS automatically creates state machines for you with built-in error handling.

There are two levels of error handling in Media Insights on AWS workflows state machines: operator error handling and workflow error handling.

## Operator error handling

There are two variations of operator error handling: operator Lambda code error handling and operator state machine Amazon States Language (ASL) error handling.

**Operator Lambda code error handling**

Operator Lambda functions can use the MasExecutionError property from the MediaInsightsEngineLambdaHelper Python library to consistently handle errors that occur within the Lambda code of Media Insights on AWS operators.

The following is an example of a Lambda function error handling used in the **Entities** (Comprehend) operator:

```
from MediaInsightsEngineLambdaHelper import MasExecutionError

try:
    ...
except Exception as e:
```

```
    operator_object.update_workflow_status("Error")
    operator_object.add_workflow_metadata(comprehend_entity_job_id=job_id,
comprehend_error="comprehend returned as failed:
{e}".format(e=response["EntitiesDetectionJobPropertiesList"][0]["Message"]))
    raise MasExecutionError(operator_object.return_output_object())
```

This code updates the outputs of the operator within the workflow_execution results with the error status, specific error information for this failure then raises an exception. The exception will invoke the Catch and Retry error handling within the state machine. Refer to the Operator state machine ASL error handling section.

**Operator state machine ASL error handling**

Operators use `Catch` and `Retry` to handle errors that occur in the steps of the operator state machine tasks. If a step returns an error, the operator is retried. If retry attempts fail, then the `OperatorFailed` Lambda resource is invoked to handle the error. Once invoked, the `OperatorFailed` Lambda resource makes sure the `workflow_execution` object contains the error status, specific information about the failure, and the workflow run error status is propagated to the control plane. The following is an example of the `Catch` and `Retry` states using ASL for Media Insights on AWS state machine error handling:

```
{
    ...
    "Retry": [ {
        "ErrorEquals": ["Lambda.ServiceException", "Lambda.AWSLambdaException",
"Lambda.SdkClientException", "Lambda.Unknown", "MasExecutionError"],
        "IntervalSeconds": 2,
        "MaxAttempts": 2,
        "BackoffRate": 2
    }],
    "Catch": [{
        "ErrorEquals": ["States.ALL"],
        "Next": <"OPERATION_NAME> Failed (<STAGE_NAME>)",
        "ResultPath": "$.Outputs"
        }]
    ...
```

# Workflow state machine error handling

If you need to perform certain actions in response to workflow errors, then edit the `WorkflowErrorHandlerLambda` Lambda function. This function is invoked when the Step

Functions service emits `Step Functions Execution Status Change` EventBridge events that have an error status (FAILED, TIMED_OUT, ABORTED). The error handler propagates the error to the Media Insights on AWS control plane if the workflow is not already completed.

If an error occurs in the Step Functions service that causes the state machine for an Media Insights on AWS workflow to be stopped immediately, then the `Catch`, `Retry`, and `OperatorFailed` Lambda functions are unable to handle the error. These types of errors can occur in circumstances such as when the Step Functions history limit is exceeded, or the Step Functions has been manually stopped. Failure to handle these errors will put the workflow in a perpetually `Started` status in the Media Insights on AWS control plane. If this happens, then you must manually remove the workflow from the `WorkflowExecution` DynamoDB table.

# Problem: Amazon CloudWatch Events bus permissions error

If during spoke stack deployment, you received a `CREATE_FAILED` message for the `TAWarnRule` or the `TASErrorRule`, verify that the CloudWatch Event Bus in the primary account allows the spoke account to send events to the primary account.

## Resolution

Update the primary stack with the secondary account ID or complete the following procedure.

1. In the primary account, navigate to the [CloudWatch console](https://).
2. In the navigation pane, select **Event Buses**.
3. Select **Add Permissions**.
4. For **Principal**, enter the applicable secondary account ID.
5. Select the **Everybody(*)** checkbox.
6. Choose **Add**.

# Contact AWS Support

If you have [AWS Developer Support](https://), [AWS Business Support](https://), or [AWS Enterprise Support](https://), you can use the Support Center to get expert assistance with this solution. The following sections provide instructions.

# Create case

1. Sign in to [Support Center](#).

2. Choose **Create case**.

# How can we help?

1. Choose **Technical**.

2. For **Service**, select **Solutions**.

3. For **Category**, select **Other Solutions**.

4. For **Severity**, select the option that best matches your use case.

5. When you enter the **Service**, **Category**, and **Severity**, the interface populates links to common troubleshooting questions. If you can't resolve your question with these links, choose **Next step: Additional information**.

# Additional information

1. For **Subject**, enter text summarizing your question or issue.

2. For **Description**, describe the issue in detail.

3. Choose **Attach files**.

4. Attach the information that AWS Support needs to process the request.

# Help us resolve your case faster

1. Enter the requested information.

2. Choose **Next step: Solve now or contact us**.

# Solve now or contact us

1. Review the **Solve now** solutions.

2. If you can't resolve your issue with these solutions, choose **Contact us**, enter the requested information, and choose **Submit**.

# Uninstall the solution

You can uninstall the Media Insights on AWS solution from the AWS Management Console or by using the AWS Command Line Interface. You must manually delete the `Dataplane` and `DataplaneLogs` S3 buckets created by this solution. AWS Solutions implementations do not automatically delete `Dataplane` and `DataplaneLogs` S3 buckets if you have stored data to retain.

## Using the AWS Management Console

1. Sign in to the AWS CloudFormation console.
2. On the **Stacks** page, select this solution's installation stack.
3. Choose **Delete**.

## Using AWS Command Line Interface

Determine whether the AWS Command Line Interface (AWS CLI) is available in your environment. For installation instructions, see What Is the AWS Command Line Interface in the *AWS CLI User Guide*. After confirming that the AWS CLI is available, run the following command:

```
$ aws cloudformation delete-stack --stack-name <installation-stack-name>
```

## Deleting the Amazon S3 buckets

This solution is configured to retain the solution-created Amazon S3 bucket (for deploying in an opt-in Region) if you decide to delete the AWS CloudFormation stack to prevent accidental data loss. After uninstalling the solution, you can manually delete this Amazon S3 bucket if you do not need to retain the data. Follow these steps to delete the Amazon S3 bucket.

1. Sign in to the Amazon S3 console.
2. Choose **Buckets** from the left navigation pane.
3. Locate the `Dataplane` S3 bucket.
4. Select the S3 bucket and choose **Empty**, then choose **Delete**.
5. Locate the `DataplaneLogs` S3 bucket.

6. Select the S3 bucket and choose **Empty**, then choose  **Delete**.

To delete the S3 bucket using AWS CLI, run the following command:

```
$ aws s3 rb s3://<bucket-name> --force
```

# Developer guide

The developer guide section provides information about this solution's source code, customizations, and API reference documentation.

# Source code

Visit our GitHub repository to download the source files for this solution and to share your customizations with others.

The Media Insights on AWS templates are generated using the AWS Cloud Development Kit (AWS CDK). Refer to the README.md file for additional information.

# Customization guide

## Implementing a new operator in Media Insights on AWS

Operators are Lambda functions that derive new media objects from input media and/or generate metadata by analyzing input media. They run as part of a Media Insights on AWS workflow. Workflows are AWS Step Functions that define the order in which operators run.

Operators can be synchronous (sync) *or* asynchronous (async). Sync operators start an analysis (or transformation) job and get its result in a single Lambda function. Async operators use separate Lambda functions to start jobs and get their results. Typically, async operators run for several minutes.

Operator inputs can include a list of media, metadata, and user-defined workflow and/or operator configurations.

Operator outputs include the run status, and Amazon S3 locations for the newly derived media and metadata objects saved in Amazon S3. These outputs get passed to other operators in downstream workflow stages.

### Step 1: Write operator Lambda functions

*Time to complete: >1 hour*

Operators reside under `source/operators`. Create a new folder there for your new operator. Copy `source/operators/rekognition/generic_data_lookup.py` to that new directory and change it to do what you require.

The Media Insights on AWS helper library must be used inside an operator to interact with the control plane and data plane. This library lives under `lib/MediaInsightsEngineLambdaHelper/`.

## Using the Media Insights on AWS helper library

Instantiate the helper:

```
from MediaInsightsEngineLambdaHelper import OutputHelper
output_object = OutputHelper("<OPERATOR-NAME>")
```

## Get asset and workflow IDs

To make it easier to find results and know the provenance of data, operators should save the files that they generate to a directory path that is unique to the workflow run ID and asset ID (for example, `s3// + dataplane_bucket + /private/assets/ + asset_id + /workflows/ + workflow_id + /`). Obtain the workflow and asset IDs from the Lambda function's entry point event object:

```
# Lambda function entrypoint:
def lambda_handler(event, context):
    workflow_id = str(event["WorkflowExecutionId"])
    asset_id = event['AssetId']
```

## Get input media objects

Media objects are passed using their location in Amazon S3. Use the boto3 Amazon S3 client to access them from Amazon S3 using the locations specified in the Lambda function's entry point event object:

```
def lambda_handler(event, context):
    if "Video" in event["Input"]["Media"]:
        s3bucket = event["Input"]["Media"]["Video"]["S3Bucket"]
        s3key = event["Input"]["Media"]["Video"]["S3Key"]
    elif "Image" in event["Input"]["Media"]:
        s3bucket = event["Input"]["Media"]["Image"]["S3Bucket"]
```

```
        s3key = event["Input"]["Media"]["Image"]["S3Key"]
```

## Get operator configuration input

Operator configurations can be accessed from the Configuration attribute in the Lambda function's entry point event object. For example, here's how the face search operator gets the user-specified face collection ID:

```
collection_id = event["Configuration"]["CollectionId"]
```

## Write data to downstream operators

Metadata derived by an operator can be passed as an input to the next stage in a workflow by adding specified data to the operator's output_object. Do this with the add_workflow_metadata function in the OutputHelper, as shown below:

> ⚠️ **Important**
>
> The values for attributes must be strings. The values for attributes must not be empty strings.

```
from MediaInsightsEngineLambdaHelper import OutputHelper
output_object = OutputHelper("<OPERATOR-NAME>")

def lambda_handler(event, context):
    ...
    # Passing MyData objects to downstream operators
    output_object.add_workflow_metadata(MyData1=my_data_1)
    output_object.add_workflow_metadata(MyData2=my_data_2)
    # Multiple key value pairs can also be specified as a list, like this:
    output_object.add_workflow_metadata(MyData3=my_data_3, MyData4=my_data_4)
    ...
    return output_object.return_output_object()
```

## Read data from upstream operators

Metadata that was output by upstream operators can be accessed from the Lambda function's entry point event object:

```
my_data_1 = event["Input"]["MetaData"]["MyData1"]
```

## Store media metadata to the data plane

Use `store_asset_metadata()` to store results. For paged results, call that function for each page:

```
from MediaInsightsEngineLambdaHelper import DataPlane
dataplane = DataPlane()
metadata_upload = dataplane.store_asset_metadata(asset_id, operator_name, workflow_id,
  response)
```

## Store media objects to the data plane

Operators can derive new media objects. For example, the Amazon Transcribe operator derives a new text object from an input audio object. Save new media objects with `add_media_object()` as shown in the code:

```
from MediaInsightsEngineLambdaHelper import MediaInsightsOperationHelper
operator_object = MediaInsightsOperationHelper(event)
operator_object.add_media_object(my_media_type, bucket, key)
The my_media_type variable should be "Video", "Audio", or "Text".
```

## Retrieve media objects from the data plane

```
from MediaInsightsEngineLambdaHelper import MediaInsightsOperationHelper
operator_object = MediaInsightsOperationHelper(event)
bucket = operator_object.input["Media"][my_media_type]["S3Bucket"]
key = operator_object.input["Media"][my_media_type]["S3Key"]
s3_response = s3.get_object(Bucket=bucket, Key=key)
Again, the my_media_type variable should be "Video", "Audio", or "Text".
```

## Step 2: Add your operator to the Media Insights on AWS operator library

*Time to complete: 30 minutes*

This step involves editing the AWS CDK source for deploying the Media Insights on AWS operator library, located at [source/cdk/lib/operator-library.ts](source/cdk/lib/operator-library.ts). You must add new entries for your operator for the following resources:

- Lambda functions

- IAM roles

- Register as operators in the control plane

- Export operator names as outputs

## Create the IAM role resource

Create an IAM resource to give your Lambda function the appropriate permissions. Media Insights on AWS operators need `AWSLambdaBasicExecutionRole` plus policies for any other AWS resource and services accessed by the Lambda function.

## Create Lambda function resource

Create a Lambda function resource for your operator Lambda function using the createLambdaFunction helper function.

The parameters of the Lambda function are listed in the following table.

| Parameter | Description |
| --- | --- |
| `id` | Logical ID of the CloudFormation template resource. |
| `codeArchive` | Name of the compressed Lambda function source files generated in build-s3-dist.sh. |
| `timeout` | Number of seconds which the Lambda function times out. |
| `props.handler` | Name of the method in Lambda function to process events. |
| `props.role` | IAM resource created in the previous step. |
| `props.tracing` | Tracing settings for Lambda function. |
| `props.memorySize` | Size of memory to allocate to Lambda function. |

| Parameter | Description |
| --- | --- |
| props.environment | Environment variables for Lambda function. |

Usage example:

```
const customOperator = createLambdaFunction(
    this,
    'customOperator',
    "custom_operator.zip",
    300,
    {
        handler: " custom_operator.lambda_handler",
        role: customOperatorLambdaRole,
        tracing: lambda.Tracing.PASS_THROUGH,
        environment: {
            OPERATOR_NAME: "CustomOperator",
            DataplaneEndpoint,
            DataLookupRole: genericDataLookupLambdaRole.roleArn,
            botoConfig
        }
    }
);
```

## Create the Media Insights on AWS operator resource using your Lambda function(s) and export the operator as output

The createCustomResource helper Lambda function can be used to create both the custom resource and the output for the new custom operator.

The parameters of the Lambda function are listed in the following table.

| Parameter | Description |
| --- | --- |
| id | Logical ID of the CloudFormation template resource. |
| props.Name | Specify the name of the custom resource. |

| Parameter | Description |
|---|---|
| props.Description | Specify the description of the custom resource. |
| props.Async | Specify whether your operator is sync or async. |
| props.Configuration | Specify the MediaType and Enabled fields and add any other configurations needed. |
| props.StartLambdaArn | Specify the ARN of the Lambda function to start your operator. |
| props.MonitorLAmbdaArn | If your operator is async, specify the ARN of the monitoring Lambda function. |
| props.OutputName | Specify the OutputName of the operator. Same as id if not specified. |
| props.ExportName | Specify the ExportName of the operator. Same as props.Name if not specified. |

Usage example:

```
createCustomResource(
    this,
    'customOperatorOperation',
    {
        Name: "customOperator",
        Description: "Operation name of CustomOperator ",
        Async: false,
        Configuration: {
            MediaType: "Video",
            Enabled: false
        },
        StartLambdaArn: startGenericDataLookup.functionArn,
        OutputName: "CustomOperator",
        ExportName: "CustomOperator"
    }
```

```
);
```

## Step 3: Update the build script to deploy your operator to AWS Lambda

Time to complete: 5 minutes

Update the `Make lambda package` section in [build-s3-dist.sh](build-s3-dist.sh) to zip your operator's Lambda function(s) into the regional distribution directory:

```
# Add operator code to a zip package for AWS Lambda
zip my_operator.zip my_operator.py
# Copy that zip to the regional distribution directory.
cp "./dist/my_operator.zip" "$regional_dist_dir/ "
```

## Step 4: Deploy your custom build

Run the build script to generate CloudFormation templates, then deploy them as described in the [README.md](README.md) file.

## Step 5: Test your new workflow and operator

To test workflows and operators, submit requests to the workflow API endpoint using IAM authorization. Tools like [Postman](Postman) (described in the [README.md](README.md) file) and [awscurl](awscurl) simplify IAM authorization. The following examples assume your AWS access key and secret key are set up correctly in `awscurl`:

Sample command to list all available workflows:

```
awscurl "$WORKFLOW_API_ENDPOINT"/workflow | cut -f 2 -d "'" | jq '.[].Name'
```

Sample command to list all stages in a workflow:

```
WORKFLOW_NAME="CasImageWorkflow"
awscurl "$WORKFLOW_API_ENDPOINT"/workflow/"$WORKFLOW_NAME" | cut -f 2 -d "'" | jq -c
  '.Stages | .[].Name'
```

Sample command to get the workflow configuration for a stage:

```
WORKFLOW_NAME="CasImageWorkflow"
STAGE_NAME="RekognitionStage"
```

```
awscurl "$WORKFLOW_API_ENDPOINT"/workflow/"$WORKFLOW_NAME" | cut -f 2 -d "'" | jq -c
  '.Stages."$STAGE_NAME".Configuration'
```

Sample command to run a workflow with the default configuration:

```
WORKFLOW_NAME="CasImageWorkflow"
aws s3 cp test_image.jpg s3://"$DATAPLANE_BUCKET"/
awscurl -X POST --data '{"Name":"$WORKFLOW_NAME", "Input":{"Media":{"Image":
{"S3Bucket":"'$DATAPLANE_BUCKET'","S3Key":"test_image.jpg"}}}}' $WORKFLOW_API_ENDPOINT/
workflow/execution
```

Sample command to run a workflow with a non-default configuration:

```
WORKFLOW_NAME="CasImageWorkflow"
CONFIGURATION='{"RekognitionStage":{"faceDetectionImage":
{"MediaType":"Image","Enabled":false},"celebrityRecognitionImage":
{"MediaType":"Image","Enabled":false},"faceSearchImage":
{"MediaType":"Image","Enabled":false},"contentModerationImage":
{"MediaType":"Image","Enabled":false},"labelDetectionImage":
{"MediaType":"Image","Enabled":false}}}'
aws s3 cp test_image.jpg s3://"$DATAPLANE_BUCKET"/
awscurl -X POST --data '{"Name":"'$WORKFLOW_NAME'", "Configuration":'$CONFIGURATION',
 "Input":{"Media":{"Image":
{"S3Bucket":"'$DATAPLANE_BUCKET'","S3Key":"test_image.jpg"}}}}' $WORKFLOW_API_ENDPOINT/
workflow/execution
```

## Monitor your test

You can monitor workflows with the following logs:

- Your operator Lambda function. To find this log, search the Lambda functions for your operator
  name.
- The data plane API Lambda function. To find this log, search Lambda functions for
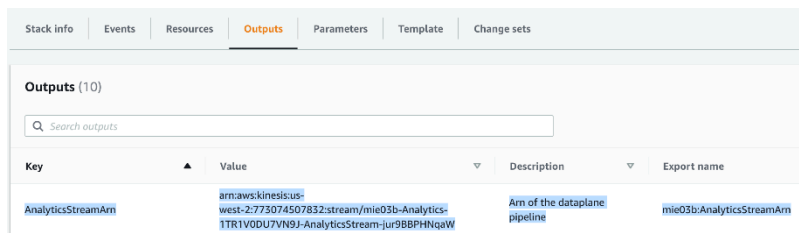  `MediaInsightsDataplaneApiStack`.

## Validate metadata in the data plane

When your operator finishes successfully, then you can refer to data saved from
the `Dataplane.store_asset_metadata()` function in the `DataPlaneTable` in Amazon
DynamoDB.

# Implementing a new data stream consumer

The data plane stores each item as an object in Amazon S3 and stores their Amazon S3 object identifier in DynamoDB. However, many application scenarios involve data access patterns that require capabilities beyond those provided by DynamoDB and Amazon S3. For example, you might need Amazon OpenSearch Service to support interactive analytics, Amazon SNS to provide real-time messaging and notifications, or Amazon QuickSight to support analytical reporting over big datasets.

The data plane provides a change-data-capture (CDC) stream from DynamoDB to communicate media analysis data to stream consumers, where ETL tasks can transform and load raw data to the downstream data stores that support end-user applications. This CDC stream is provided as a Kinesis Data Stream. The ARN for this is provided as an output called AnalyticsStreamArn in the base Media Insights on AWS CloudFormation stack, as shown below:



For more information about how to implement Kinesis data stream consumers in Media Insights on AWS, refer to the Media Insights on AWS demo application, which includes a data stream consumer that feeds Amazon OpenSeach Service.

# API reference

**Data plane API**

- `GET /`
- `POST /create`
- `POST /download`
- `GET /mediapath/{asset_id}/{workflow_id}`
- `GET /metadata`
- `DELETE /metadata/{asset_id}`
- `DELETE /metadata/{asset_id}/{operator_name}`
- `POST /upload`

**Workflow (control plane) API**

- GET /

- POST /system/configuration

- POST /workflow

- GET /workflow/configuration/{Name}

- POST /workflow/execution

- GET /workflow/execution/asset/{AssetId}

- GET /workflow/execution/status/{Status}

- DELETE /workflow/execution/{Id}

- GET /workflow/execution/{Id}

- POST /workflow/operation

- DELETE /workflow/operation/{Name}

- POST /workflow/stage

- DELETE /workflow/stage/{Name}

- DELETE /workflow/{Name}

## Data plane API

## Create an asset in the data plane from a json input composed of the input key and bucket of the object:

POST /create

**Body:**

```
{
    "Input": {
        "S3Bucket": "{somenbucket}",
        "S3Key": "{somekey}"
    }
}
```

**Returns:**

A dictionary mapping of the asset ID and the new location of the media object.

## Retrieve metadata for an asset:

GET /metadata/{asset_id}

Returns: All asset metadata. If the result provides a cursor then you can get the next page by specifying the cursor:

GET /metadata/{asset_id}?cursor={cursor}

## Add operation metadata for an asset:

POST /metadata/{asset_id}

**Body:**

```
{
    "OperatorName": "{some_operator}",
    "Results": "{json_formatted_results}"
}
```

## Retrieve the metadata that a specific operator created from an asset:

GET /metadata/{asset_id}/{operator_name}

## Get version information:

GET /version

**Returns:**

A dictionary containing the version of the Media Insights on AWS framework and the version of the data plane API. Since it is possible for the Media Insights on AWS framework to be released without any API changes, these two versions can be different. The Media Insights on AWS framework and its APIs are versioned according to Semantic Versioning rules. Under this scheme, version numbers and the way they change convey meaning about backwards compatibility.

For example, if the Media Insights on AWS framework was version v2.0.4 and the workflow API was version 2.0.0, then this would return the following response:

```
b'{"ApiVersion":"2.0.0","FrameworkVersion":"v2.0.4"}'
```

# Workflow API

## Add a new system configuration parameter or update an existing Media Insights on AWS system configuration parameter

POST /system/configuration

**Body:**

```
{
    "Name": "ParameterName",
    "Value": "ParameterValue"
}
```

Supported parameters:

**MaxConcurrentWorkflows** - Sets the maximum number of workflows that are allowed to run concurrently. Any new workflows that are added after MaxConcurrentWorkflows is reached are placed on a queue until capacity is freed by completing workflows. Use this to help avoid throttling in service API calls from workflow operators. This setting is checked each time the WorkflowSchedulerLambda is run and may take up to 60 seconds to take effect.

**Returns:**

Nothing

Raises:

- 200: The system configuration was set successfully
- 400: Bad Request
- 500: Internal server error - an input value is not valid

## Get the current Media Insights on AWS system configuration:

GET /system/configuration

**Returns:**

A list of dictionaries containing the current Media Insights on AWS system configuration key-value pairs.

Raises:

- 200: The system configuration was returned successfully
- 500: Internal server error

## Create a workflow from a list of existing stages

A workflow is a pipeline of stages that are started sequentially to transform and extract metadata for a set of MediaType objects. Each stage must contain either a *Next* key indicating the next stage to run or an *End* key indicating it is the last stage.

`POST /workflow`

**Body:**

```
{
    "Name": string,
    "StartAt": string - name of starting stage,
    "Stages": {
        "stage-name": {
            "Next": "string - name of next stage"
        },
        ...,
        "stage-name": {
            "End": true
        }
    }
}
```

**Returns:**

A dictionary mapping keys to the corresponding workflow created including the AWS resources used to run each stage.

Raises:

- 200: The workflow was created successfully
- 400: Bad Request - one of the input stages was not found or was not valid

- 500: Internal server error

## List all workflow definitions

GET /workflow

**Returns:**

A list of workflow definitions.

Raises:

- 200: All workflows returned successfully
- 500: Internal server error

## Get a workflow configuration object by name

GET /workflow/configuration/{Name}

**Returns:**

A dictionary containing the workflow configuration.

**Raises:**

- 200: All workflows returned successfully
- 404: Not found
- 500: Internal server error

## Run a workflow

The Body contains the name of the workflow to run, and at least one input media type within the media object. A dictionary of stage configuration objects can be passed in to override the default configuration of the operations within the stages.

POST /workflow/execution

**Body:**

```
{
```

```
 "Name":"Default",
 "Input": media-object
 "Configuration": {
     {
     "stage-name": {
         "Operations": {
             "SplitAudio": {
                 "Enabled": True,
                 "MediaTypes": {
                     "Video": True/False,
                     "Audio": True/False,
                     "Frame": True/False
                 }
             },
         },
     }
     ...
     }
 }
```

**Returns:**

A dictionary describing the workflow run properties and the **WorkflowExecutionId** that can be used as the Id in `/workflow/execution/{Id}` API requests.

Raises:

- 200: The workflow run was created successfully
- 400: Bad Request - the input workflow was not found or was not valid
- 500: Internal server error

## List all workflow runs

`GET /workflow/execution`

**Returns:**

A list of workflow runs.

**Raises:**

- 200: List returned successfully

- 500: Internal server error

## Get workflow runs by asset ID

`GET /workflow/execution/asset/{AssetId}`

**Returns:**

A list of dictionaries containing the workflow runs matching the AssetId.

**Raises:**

- 200: List returned successfully
- 404: Not found
- 500: Internal server error

## Get all workflow runs with the specified status

`GET /workflow/execution/status/{Status}`

**Returns:**

A list of dictionaries containing the workflow runs with the requested status.

**Raises:**

- 200: All workflows returned successfully
- 404: Not found
- 500: Internal server error

## Delete a workflow run

`DELETE /workflow/execution/{Id}`

**Returns:**

Nothing.

**Raises:**

- 200: Workflow run deleted successfully

- 404: Not found

- 500: Internal server error

## Get a workflow run by ID

```
GET /workflow/execution/{Id}
```

**Returns:**

A dictionary containing the workflow run.

**Raises:**

- 200: The workflow run details returned successfully

- 404: Not found

- 500: Internal server error

## Create a new operation

```
POST /workflow/operation
```

Generates an operation state machine using the operation Lambda function(s) provided.

Creates a singleton operator stage that can be used to run the operator as a single-operator stage in a workflow.

Operators can be *sync* or *async*. Sync operators complete before returning control to the invoker, while async operators return control to the invoker immediately after the operation is successfully initiated. Async operators require an additional monitoring task to check the status of the operation.

For more information on how to implement Lambda functions to be used in Media Insights on AWS operators, refer to [Implementing a new operator in Media Insights on AWS](#).

**Body:**

```
{
    "Name":"operation-name",
    "Type": ["Async"|"Sync"],
```

```
    "Configuration" : {
            "MediaType": "Video",
            "Enabled:": True,
            "configuration1": "value1",
            "configuration2": "value2",
            ...
        }
    "StartLambdaArn":arn,
    "MonitorLambdaArn":arn,
    "SfnExecutionRole": arn
    }
```

**Returns:**

A dictionary mapping keys to the corresponding operation:

```
{
    "Name": string,
    "Type": ["Async"|"Sync"],
    "Configuration" : {
        "MediaType": "Video|Frame|Audio|Text|...",
        "Enabled:": boolean,
        "configuration1": "value1",
        "configuration2": "value2",
        ...
    }
    "StartLambdaArn":arn,
    "MonitorLambdaArn":arn,
    "StateMachineExecutionRoleArn": arn,
    "StateMachineAsl": ASL-string
    "StageName": string
}
```

Raises:

- 200: The operation and stage was created successfully
- 400: Bad Request


- one of the input lambdas was not found
- one or more of the required input keys is missing
- an input value is not valid

- 409: Conflict

- 500: Internal server error

> ⚠️ **Important**
>
> Do not try to create more than 35 new operators via /workflow/operation. The IAM
> inline policy used in media-insights-stack.yaml to grant **InvokeFunction** permission to
> the **StepFunctionRole** for new operators will exceed the maximum length allowed by IAM
> if users create more than 35 operators (+/- 1).

For more information, refer to the comments in this commit: [aws-solutions/media-insights-on-aws@451ec2e](#).

Sample command that shows how to create an operator from `/workflow/operation` on the
command line:

```
OPERATOR_NAME="op1"
WORKFLOW_API_ENDPOINT="https://tvplry8vn3.execute-api.us-west-2.amazonaws.com/api/"
START_ARN="arn:aws:lambda:us-west-2:773074507832:function:mie03d-
OperatorFailedLambda-11W1LAY0CWCUZ"
MONITOR_ARN="arn:aws:lambda:us-west-2:773074507832:function:mie03d-
OperatorFailedLambda-11W1LAY0CWCUZ"
REGION="us-west-2"
awscurl --region ${REGION} -X POST -H "Content-Type: application/json" -d
 '{"StartLambdaArn": "'${START_ARN}'"", "Configuration": {"MediaType": "Video",
 "Enabled": true}, "Type": "Async", "Name": "'${OPERATOR_NAME}'", "MonitorLambdaArn":
 "'${MONITOR_ARN}}'"' ${WORKFLOW_API_ENDPOINT}workflow/operation;
```

## List all defined operators

`GET /workflow/operation`

**Returns:**

A list of operation definitions.

**Raises:**

- 200: All operations returned successfully

- 500: Internal server error

## Delete an operation

```
DELETE /workflow/operation/{Name}
```

**Raises:**

- 200: Operation deleted successfully
- 500: Internal server error

## Get an operation definition by name

```
GET /workflow/operation/{Name}
```

**Returns:**

A dictionary containing the operation definition.

Raises:

- 200: All operations returned successfully
- 404: Not found
- 500: Internal server error

## Create a stage state machine from a list of existing operations

A stage is a set of operations that are grouped so they can be run in parallel. When the stage is run as part of a workflow, operations within a stage are run as branches in a parallel AWS Step Functions state. The generated state machines status is tracked by the workflow engine control plane during the run.

An optional configuration for each operator in the stage can be input to override the default configuration for the stage.

```
POST /workflow/stage
```

```
{
 "Name":"stage-name",
```

```
 "Operations": ["operation-name1", "operation-name2", ...]
 }
Returns:
A dict mapping keys to the corresponding stage created including the ARN of the state
 machine created.

{
"Name": string, "Operations": [

"operation-name1", "operation-name2", …
], "Configuration": {

"operation-name1": operation-configuration-object1, "operation-name2": operation-
configuration-object1, …
} "StateMachineArn": ARN-string

"Name": "TestStage", "Operations": [

"TestOperator"
], "Configuration": {

"TestOperator": {
"MediaType": "Video", "Enabled": true
}
}, "StateMachineArn": "arn:aws:states:us-west-2:526662735483:stateMachine:TestStage"
}
```

**Raises:**

- 200: The stage was created successfully

- 400: Bad Request - one of the input state machines was not found or was not valid

- 409: Conflict

- 500: Internal server error

## List all stage definitions

```
GET /workflow/stage
```

**Returns:**

A list of operation definitions.

**Raises:**

- 200: All operations returned successfully

- 500: Internal server error

## Delete a stage

```
DELETE /workflow/stage/{Name}
```

Returns:

Nothing.

Raises:

- 200: Stage deleted successfully

- 404: Not found

- 500: Internal server error

## Get a stage definition by name

```
GET /workflow/stage/{Name}
```

**Returns:**

A dictionary containing the stage definition.

**Raises:**

- 200: Stage definition was returned successfully

- 404: Not found

- 500: Internal server error

## Delete a workflow

```
DELETE /workflow/{Name}
```

**Returns:**

Nothing.

**Raises:**

- 200: Workflow deleted successfully
- 404: Not found
- 500: Internal server error

## Get a workflow definition by name

```
GET /workflow/{Name}
```

**Returns:**

A dictionary containing the workflow definition.

**Raises:**

- 200: Workflow definition returned successfully
- 404: Not found
- 500: Internal server error

## Get version information

```
GET /version
```

**Returns:**

A dictionary containing the version of the Media Insights on AWS framework and the version of the workflow API. Since it is possible for the Media Insights on AWS framework to be released without any API changes, these two versions can be different. The Media Insights on AWS framework and its APIs are versioned according to Semantic Versioning rules. Under this scheme, version numbers and the way they change convey meaning about backwards compatibility.

For example, if the Media Insights on AWS framework was version v2.0.4 and the workflow API was version 2.0.0, then this would return the following response:

```
b'{"ApiVersion":"2.0.0","FrameworkVersion":"v2.0.4"}'
```

# Reference

This section includes information about an optional feature for collecting unique metrics for this solution, pointers to related resources, and a list of builders who contributed to this solution.

## Anonymized data collection

This solution includes an option to send anonymized operational metrics to AWS. We use this data to better understand how customers use this solution and related services and products. When invoked, the following information is collected and sent to AWS:

- **Solution ID** - The AWS solution identifier
- **Unique ID (UUID)** - Randomly generated, unique identifier for each Media Insights on AWS deployment
- **Timestamp** - Data-collection timestamp
- **Example: Instance Data** - Count of the state and type of instances that are managed by the Amazon Elastic Compute Cloud (Amazon EC2) Scheduler in each AWS Region

Example data:

Running: {`t2.micro:2`}, {`m3.large:2`}

Stopped: {`t2.large:1`}, {`m3.xlarge:3`}

AWS owns the data gathered though this survey. Data collection is subject to the [AWS Privacy Notice](). To opt out of this feature, complete the following steps before launching the AWS CloudFormation template.

1. Download the AWS CloudFormation template to your local hard drive.
2. Open the AWS CloudFormation template with a text editor.
3. Modify the AWS CloudFormation template mapping section from:

```
AnonymizedData:
    SendAnonymizedData:
        Data: Yes
```

    to:

```
AnonymizedData:
    SendAnonymizedData:
        Data: No
```

4. Sign in to the AWS CloudFormation console.

5. Select **Create stack**.

6. On the **Create stack** page, specify template section, select **Upload a template file**.

7. Under **Upload a template file**, choose **Choose file** and select the edited template from your local drive.

8. Choose **Next** and follow the steps in Launch the stack in the Deploy the solution section of this guide.

# Related resources

Content Localization on AWS, a solution built on the Media Insights on AWS framework, helps extend the reach of your video-on-demand (VOD) content by efficiently creating accurate multi-language subtitles using AWS AI services. You can make manual corrections to the automatically created subtitles and use advanced AWS AI service customization features to improve the results of the automation for your content.

# Contributors

- Alex Burkleaux
- Brandon Dold
- Ian Downard
- Jill Adams
- David Chung
- Colin McCoy
- Eric Thoman

# Revisions

| Date | Change |
|---|---|
| April 2021 | Initial release (v2.0.5) |
| July 2021 | Release v3.0.1: Updated AWS CloudFormation template parameters for capturing operational metrics. For more information, refer to the CHANGELOG.md file in the GitHub repository. |
| August 2021 | Release v3.0.2: Bug fixes. For more information, refer to the CHANGELOG.md file in the GitHub repository. |
| November 2021 | Release v3.0.4: Bug fixes. For more information, refer to the CHANGELOG.md file in the GitHub repository. |
| January 2022 | Release v4.0.0: Added support for custom language models in Amazon Transcribe and source language auto-detection in Amazon Translate. For more information, refer to the CHANGELOG.md file in the GitHub repository. |
| February 2022 | Release v4.0.1: Bug fixes. For more information, refer to the CHANGELOG.md file in the GitHub repository. |
| April 2022 | Release v5.0.0: Added customer managed keys for enhanced security. For more information, refer to the CHANGELOG.md file in the GitHub repository. |
| January 2023 | Release v5.0.1: Updated Python libraries for enhanced security. For more information, refer |

| Date | Change |
| --- | --- |
|  | to the CHANGELOG.md file in the GitHub repository. |
| February 2023 | Release v5.1.0: Added AppRegistry support to register the CloudFormation stack and the underlying resources as an application in both Service Catalog AppRegistry and AWS Systems Manager Application Manager. For more information, refer to the CHANGELOG.md file in the GitHub repository. |
| April 2023 | Release v5.1.1: Mitigated impact caused by new default settings for S3 Object Ownership (ACLs disabled) for all new S3 buckets. For more information, refer to the CHANGELOG.md file in the GitHub repository. |
| April 2023 | Release v5.1.2: Internal bug fix only. For more information, refer to the CHANGELOG.md file in the GitHub repository. |
| May 2023 | Release v5.1.3: Added `package-lock.json` files for resources that uses Node.js and removed GitHub Actions workflow files. For more information, refer to the CHANGELOG.md file in the GitHub repository. |
| September 2023 | Release v5.1.4: Updated parameter names for consistency. Updated build specifications to Node.js 18 and Python to v3.11. For more information, refer to the CHANGELOG.md file in the GitHub repository. |

| Date | Change |
| --- | --- |
| November 2023 | Release v5.1.5: Updated package versions to resolve security vulnerabilities. For more information, refer to the CHANGELOG.md file in the GitHub repository. |
| November 2023 | Documentation update: Added Confirm cost tags associated with the solution to the Monitoring the solution with AWS Service Catalog AppRegistry section. |
| March 2024 | Release v5.1.6: Removed duplicate Amazon SQS policy. Specified chalice version in build script. For more information, refer to the CHANGELOG.md file in the GitHub repository. |
| June 2024 | Release v5.1.7: Upgraded vulnerable packages. For more information, refer to the CHANGELOG.md file in the GitHub repository. |
| August 2024 | Release v5.1.8: Upgraded vulnerable packages. For more information, refer to the CHANGELOG.md file in the GitHub repository. |
| November 2024 | Release v5.1.9: Upgraded vulnerable packages. For more information, refer to the CHANGELOG.md file in the GitHub repository. |

# Notices

Customers are responsible for making their own independent assessment of the information in this document. This document: (a) is for informational purposes only, (b) represents AWS current product offerings and practices, which are subject to change without notice, and (c) does not create any commitments or assurances from AWS and its affiliates, suppliers or licensors. AWS products or services are provided "as is" without warranties, representations, or conditions of any kind, whether express or implied. AWS responsibilities and liabilities to its customers are controlled by AWS agreements, and this document is not part of, nor does it modify, any agreement between AWS and its customers.

Media Insights on AWS is licensed under the terms of the Apache License Version 2.0.