



22/04/2024

**END-TO-END MULTI-FORMAT VIDEO PROCESSING AND
SECURE DELIVERY: DOCUMENTATION AND
IMPLEMENTATION GUIDE**

Created By : HEMANTHRAJ J



TABLE OF CONTENTS

TITLE	PAGE.NO
CHAPTER I	6
INTRODUCTION	6
PROBLEM STATEMENT:	6
OBJECTIVES:	6
PROJECT DESCRIPTION	6
PROJECT GOALS	7
PROJECT DELIVERABLES	7
CHAPTER II	8
SYSTEM REQUIREMENTS	8
HARDWARE REQUIREMENTS:	8
SOFTWARE REQUIREMENTS:	8
CHAPTER III	10
WORKFLOW	10
ARCHITECTURE:	10
COMPONENTS:	10
DATA FLOW:	10
SYSTEM INTERACTIONS:	10
TRIGGERED EVENTS:	10
SERVICE INTEGRATIONS:	11
USER INTERACTION:	11
CHAPTER IV	12
MODULES	12
DEVELOPMENT ENVIRONMENT SETUP:	12
AWS ACCOUNT SETUP:	12
LOCAL DEVELOPMENT ENVIRONMENT:	12
AWS SDK AND CLI:	12
LOCAL TESTING AND EMULATION:	12
DYNAMODB INTEGRATION:	12
BACKEND DEVELOPMENT WITH PHP:	13
MEDIA CONVERT:	13
JOBS IN MEDIA CONVERT:	13
SUPPORTED FORMATS IN MEDIA CONVERT:	13
VIDEO CODECS:	14

AUDIO CODECS:	14
CONTAINER FORMATS:	14
MEDIAPACKAGE:	15
COMPONENTS OF MEDIA PACKAGE:	15
PACKAGING GROUPS:	15
PACKAING CONFIGURATIONS:	15
ASSETS:	16
ENDPOINTS:	16
CLOUDFRONT DISTRIBUTION:	16
ORIGIN ENDPOINTS:	16
CHANNEL:	16
IAM ROLE:	17
HOW IAM ROLE USED HERE:	17
LAMBDA EXECUTION ROLE:	17
MEDIACONVERT ROLE:	17
MEDIAPACKAGE ROLE:	17
AWS SECRETS MANAGER ROLE:	18
CLOUDFRONT ROLE:	18
DYNAMODB ROLE:	18
CLOUDWATCH:	18
HOW CLOUDWATCH USED HERE:	18
LAMBDA FUNCTION LOGGING:	18
MONITORING LAMBDA METRICS:	19
S3 EVENT LOGGING:	19
MEDIACONVERT AND MEDIAPACKAGE LOGGING:	19
CLOUDFRONT ACCESS LOGS:	19
MONITORING DYNAMODB:	19
HOW TO MONITOR LOGS FROM LAMBDA FUNCTIONS:	20
S3 BUCKET:	20
CREATE AN BUCKET TO STORE THE INPUT FILES IN S3:	21
LAMBDA:	22
CREATE AN IAM ROLE FOR LAMBDA FUNCTION:	23
UPDATE LAMBDA FUNCTION:	26
CREATE AN IAM ROLE FOR MEDIACONVERT:	26
CREATE A FUNCTION IN AWS LAMBDA FOR MEDIACONVERT:	27
LAMBDA INTEGRATION WITH S3 BUCKET:	28

CONFIGURE THE LAMBDA CODE EDITOR TO EXECUTE MEDIA CONVERT:	30
lambda_function.py	30
job.json	32
TEST THE LAMBDA FUNCTION:	39
UPDATE THE EXISTING LAMBDA IAM ROLE FOR MEDIAPACKAGE:	40
ALLOWING AWS ELEMENTAL MEDIAPACKAGE TO ACCESS OTHER AWS SERVICES	42
Step 1: CREATE A POLICY	42
CLOUDFRONT ACCESS FOR MEDIAPACKAGE:	43
AMAZON S3 ACCESS FOR VOD WORKFLOWS:	43
SECRETS MANAGER ACCESS FOR CDN AUTHORIZATION:	44
Step 2: CREATE A ROLE	45
CREATE AN IAM ROLE FOR MEDIA PACKAGE:	45
CREATE AN IAM ROLE FOR SECRET MANAGER:	46
Step 3: MODIFY THE TRUST RELATIONSHIP	47
CDN authorization in AWS Elemental MediaPackage:	47
Custom HTTP header and example value.	48
Secret key and example value.	48
SETTING UP CDN AUTHORIZATION:	49
Step 1: Configure a CDN custom origin HTTP header	49
Step 2: Store the value as a secret in AWS Secrets Manager	49
To store a secret in Secrets Manager	50
CREATE A TABLE IN DYNAMODB FOR STORING METADATA:	52
CREATE AN ANOTHER FUNCTION IN AWS LAMBDA FOR MEDIAPACKAGE:	54
Integrate the S3 destination bucket with a lambda function:	54
lambda_funation.py	54
CODE WALKTHROUGH:	60
TEST THE LAMBDA FUNCTION:	61
REQUIREMENTS INSTALLING	61
INSTALLING XAMPP SERVER:	61
INSTALLING COMPOSER:	62
ADDITIONAL NOTES:	62
CREATE A USER INTERFACE FOR DELIVER THE PROCESSED VIDEO CONTENT:	63
index.php	63
Create a composer.json File:	65
Install Dependencies:	66
TEST THE USER INTERFACE IN LOCAL:	66

START XAMPP:	66
PLACE PROJECT DIRECTORY:	67
ACCESS PHP FILE IN WEB BROWSER:	67
VIEW PHP OUTPUT:	68
STOPPING XAMPP:	68
CHAPTER V	68
CONCLUSION	68

CHAPTER I

INTRODUCTION

In today's digital age, the demand for efficient and scalable video processing and delivery solutions has never been greater. With the proliferation of online video content across various platforms, businesses and creators alike seek robust systems to handle the upload, conversion, storage, and secure delivery of their media assets. Our project addresses these needs by leveraging the power of cloud computing and AWS services to create a comprehensive video processing and delivery pipeline.

PROBLEM STATEMENT:

In today's digital era, the demand for high-quality video content delivery is ever-increasing. However, efficiently managing video processing, storage, and delivery while ensuring security and scalability presents significant challenges for content providers. To address these challenges, we propose the development of an Automated Video Processing and Delivery System.

OBJECTIVES:

The objective of this project is to design and implement a robust system that automates the processing and delivery of uploaded videos. The system should seamlessly handle video conversion into multiple formats with varying resolutions, ensure secure storage and delivery via a Content Delivery Network (CDN), and provide an intuitive user interface for accessing the content.

PROJECT DESCRIPTION

Our project revolves around the creation of a robust video processing and delivery system leveraging AWS services. Upon uploading a video file to an S3 bucket, a series of Lambda functions are triggered to initiate the conversion process using AWS Elemental MediaConvert. This service converts the videos into multiple formats and resolutions based on predefined configurations.

The converted videos are then securely stored in another S3 bucket. Simultaneously, a Lambda function is triggered to configure secure content delivery through Amazon CloudFront, ensuring authorized access to the videos. Additionally, integration with AWS Elemental MediaPackage facilitates optimized video ingestion into the delivery pipeline.

To provide users with an intuitive interface for accessing and viewing videos, we develop a user-friendly web application inspired by popular video-sharing platforms. This application retrieves endpoint information from a DynamoDB database, allowing users to browse and watch converted videos seamlessly.

PROJECT GOALS

The primary goals of our project include:

1. **Automation:** Reduce manual intervention in the video processing and delivery workflow to improve efficiency and scalability.
2. **Scalability:** Design a solution capable of handling varying workloads and adapting to changing demands.
3. **Security:** Ensure that video content is securely stored and delivered, preventing unauthorized access.
4. **User Experience:** Develop an intuitive interface for users to easily upload, view, and interact with video content.

PROJECT DELIVERABLES

Our project delivers a comprehensive video processing and delivery pipeline, comprising automated processes for:

1. Uploading video files to Amazon S3.
2. Triggering Lambda functions for video conversion using AWS Elemental MediaConvert.
3. Storing converted videos in designated S3 buckets.
4. Configuring secure content delivery through Amazon CloudFront.
5. Integrating with AWS Elemental MediaPackage for optimized video ingestion.
6. Developing a user-friendly interface for accessing and viewing videos.

CHAPTER II

SYSTEM REQUIREMENTS

HARDWARE REQUIREMENTS:

1. **AWS Platform Account:** Access to an active AWS account is necessary for utilizing AWS services.
2. **Internet Connectivity:** Stable internet connectivity is essential for accessing AWS services, downloading necessary tools and libraries, and interacting with the project's user interface.
3. **Memory (RAM):** Sufficient memory (RAM) is important for efficient execution of serverless functions and handling large files during document analysis.

SOFTWARE REQUIREMENTS:

1. **AWS Account:** You'll need an AWS account to utilize various cloud services such as S3, Lambda, MediaConvert, CloudFront, MediaPackage, and DynamoDB.
2. **AWS CLI:** The AWS Command Line Interface (CLI) enables you to interact with AWS services from the command line, facilitating automation and management tasks.
3. **IDE (Integrated Development Environment):** Choose an IDE that suits your preferences and supports the programming languages you'll be using for Lambda functions and web development. Popular choices include Visual Studio Code, PyCharm, and IntelliJ IDEA.
4. **Programming Languages:** Depending on your preferences and the requirements of your Lambda functions and web application, you may need languages such as Python, Node.js (JavaScript), or Java.
5. **Version Control:** Use a version control system such as Git to manage your project's source code efficiently. Platforms like GitHub, GitLab, or Bitbucket can host your repositories and provide collaboration features.
6. **Web Development Tools:** If you're building a web-based user interface, you'll need tools for frontend development such as HTML, CSS, and JavaScript frameworks/libraries like React.js, Angular, or Vue.js.
7. **Database Management Tools:** For working with DynamoDB, you may need tools like AWS Management Console or third-party DynamoDB client tools for querying and managing data.

8. Testing Frameworks: Depending on your project's requirements, you may need testing frameworks for both backend (Lambda functions) and frontend (UI). Examples include Jest for JavaScript testing, PyTest for Python testing, and Selenium for UI testing.

9. Dependency Management: Use package managers like npm (for Node.js projects) or pip (for Python projects) to manage dependencies for your Lambda functions and web application.

10. Documentation Tools: Choose tools for documenting your project, such as Markdown editors (e.g., Typora) for writing README files and project documentation.

11. Monitoring and Logging Tools: Utilize AWS CloudWatch for monitoring Lambda function logs, metrics, and alarms to ensure the health and performance of your application.

CHAPTER III

WORKFLOW

ARCHITECTURE:

COMPONENTS:

1. S3 Buckets: For video storage and converted video storage.
2. Lambda Functions: For video processing (conversion) and triggering packaging.
3. AWS MediaConvert: For video transcoding.
4. AWS Elemental MediaPackage: For video ingest and delivery.
5. CloudFront: For content delivery network (CDN) and access control.
6. DynamoDB: For storing endpoint data and related information.
7. User Interface: Front-end application for user interaction.

DATA FLOW:

1. User uploads video to S3 bucket.
2. S3 bucket triggers Lambda function for video conversion.
3. Lambda function retrieves video from S3, converts it using MediaConvert, and stores converted videos in another S3 bucket.
4. Storage of converted videos triggers another Lambda function for packaging.
5. Packaging Lambda function configures CloudFront distributions for video delivery and stores endpoint data in DynamoDB.
6. Ingested videos are made available through MediaPackage.
7. Endpoint data is retrieved from DynamoDB and displayed in the UI for user access.

SYSTEM INTERACTIONS:

TRIGGERED EVENTS:

- S3 bucket events trigger Lambda functions for video processing and packaging.

SERVICE INTEGRATIONS:

- Lambda functions interact with S3 for video retrieval and storage.
- MediaConvert is used for video transcoding.
- MediaPackage handles video ingest and delivery.
- DynamoDB stores endpoint data for retrieval in the UI.
- CloudFront provides CDN for video delivery and access control.

USER INTERACTION:

- Users interact with the system through the UI, accessing videos delivered via CloudFront.

CHAPTER IV

MODULES

DEVELOPMENT ENVIRONMENT SETUP:

AWS ACCOUNT SETUP:

- Sign up for an AWS account if you haven't already.
- Set up IAM users and roles with appropriate permissions for accessing AWS services.
- Obtain access keys and secret keys for authentication.

LOCAL DEVELOPMENT ENVIRONMENT:

- Choose an IDE or code editor for local development.
- Install PHP and necessary dependencies for PHP development.
- Set up Composer for managing PHP dependencies.
- Install Node.js and npm for managing JavaScript dependencies.

AWS SDK AND CLI:

- Install the AWS CLI for command-line interaction with AWS services.
- Install the AWS SDK for PHP for programmatic access to AWS services in your PHP backend.
- Configure the AWS CLI with access keys and region information.

LOCAL TESTING AND EMULATION:

- Set up local emulators for AWS services like DynamoDB and S3 for testing purposes.
- Install DynamoDB Local and configure it to emulate DynamoDB on your local machine.
- Use tools like LocalStack for emulating other AWS services locally if needed.
- Set up a local testing environment for your PHP backend using PHPUnit or other testing frameworks.

DYNAMODB INTEGRATION:

- Use the AWS SDK for PHP to interact with DynamoDB in your PHP backend.
- Configure DynamoDB tables and indexes as per your project requirements.
- Write code to perform CRUD operations on DynamoDB tables from your PHP backend.

BACKEND DEVELOPMENT WITH PHP:

- Write PHP code to interact with DynamoDB.
- Implement business logic for authentication and authorization using data retrieved from DynamoDB.
- Connect your PHP backend with frontend/UI components for data presentation and interaction.

MEDIA CONVERT:

AWS Elemental MediaConvert is a cloud-based service that simplifies the process of transcoding media files (such as video and audio) into various formats suitable for delivery to a wide range of devices and platforms. With MediaConvert, users can easily convert their media files from one format to another, adjust parameters like resolution, bitrate, and codec settings, and apply enhancements such as captions and watermarks.

MediaConvert provides a scalable and flexible solution for video processing, allowing users to transcode large volumes of media files quickly and efficiently. It supports a wide range of input and output formats, including popular codecs like H.264, H.265, and ProRes, as well as adaptive bitrate streaming formats like HLS and MPEG-DASH.

One of the key features of MediaConvert is its ability to handle complex transcoding workflows through the use of presets and job settings. Users can create custom presets tailored to their specific requirements or choose from a variety of built-in presets optimized for different use cases, such as streaming, broadcast, or archival.

JOBS IN MEDIA CONVERT:

Jobs represent individual media transcoding tasks submitted to AWS Elemental MediaConvert. Each job specifies the input media file, output settings (such as format, codec, bitrate, and resolution), and any additional processing options (such as captions, watermarks, or audio normalization). Jobs are processed asynchronously, allowing users to submit multiple transcoding tasks simultaneously.

SUPPORTED FORMATS IN MEDIA CONVERT:

AWS Elemental MediaConvert supports a wide range of input and output formats for media transcoding. These formats encompass video, audio, and container formats, allowing users to convert their media files into formats suitable for delivery to various devices and platforms. Here's a breakdown of the supported formats:

VIDEO CODECS:

MediaConvert supports several video codecs for encoding and decoding video content. Commonly supported codecs include:

- H.264 (Advanced Video Coding, AVC): Widely used for streaming and broadcasting high-definition video content.
- H.265 (High Efficiency Video Coding, HEVC): Provides improved compression efficiency over H.264, enabling higher quality video at lower bitrates.
- ProRes: A high-quality, visually lossless codec developed by Apple for professional video editing and production workflows.
- VP9: An open-source video codec developed by Google, known for its efficiency in delivering high-quality video over the web.

AUDIO CODECS:

MediaConvert also supports various audio codecs for encoding and decoding audio content. Some of the supported audio codecs include:

- AAC (Advanced Audio Coding): A widely used codec for compressing audio files with high-quality output at lower bitrates.
- MP3 (MPEG-1 Audio Layer III): A popular codec for compressing audio files with broad compatibility across devices and platforms.
- AC3 (Dolby Digital): Commonly used for encoding surround sound audio for DVDs, Blu-rays, and digital television broadcasts.
- PCM (Pulse-Code Modulation): Provides uncompressed audio with high fidelity, suitable for professional audio production and mastering.

CONTAINER FORMATS:

MediaConvert supports various container formats for encapsulating video, audio, and metadata into a single file. Commonly supported container formats include:

- MP4 (MPEG-4 Part 14): A widely used multimedia container format compatible with most devices and platforms, commonly used for streaming and distribution of video content.
- MOV (QuickTime Movie): Developed by Apple, MOV is a multimedia container format capable of containing multiple tracks of video, audio, and subtitles.
- HLS (HTTP Live Streaming): An adaptive bitrate streaming protocol developed by Apple for delivering live and on-demand multimedia content over the internet.

- MPEG-TS (MPEG Transport Stream): A standard container format for transmission and storage of audio, video, and data, commonly used in broadcasting and IPTV services.

MEDIAPACKAGE:

AWS Elemental MediaPackage is a comprehensive video packaging and delivery service provided by Amazon Web Services (AWS). It enables users to effortlessly package, encrypt, and deliver video content to viewers across a wide range of devices and platforms with low latency and high reliability. MediaPackage simplifies the complexities associated with video delivery, offering features such as content protection, adaptive bitrate streaming, and seamless integration with other AWS services like Amazon CloudFront and AWS Elemental MediaLive.

MediaPackage acts as a middle layer between video origination (e.g., live streaming or video-on-demand) and distribution, allowing users to ingest raw video streams and transform them into formats suitable for delivery over the internet. This transformation includes packaging the video content into industry-standard formats like HLS (HTTP Live Streaming) and DASH (Dynamic Adaptive Streaming over HTTP), as well as applying encryption for secure content delivery.

Moreover, MediaPackage integrates seamlessly with other AWS services, enabling users to build end-to-end video workflows tailored to their specific requirements. For example, users can leverage AWS Elemental MediaLive for live video encoding, AWS Elemental MediaConvert for transcoding, AWS Lambda for custom processing tasks, and Amazon CloudFront for global content delivery through edge locations worldwide.

COMPONENTS OF MEDIA PACKAGE:

AWS Elemental MediaPackage consists of several key components that work together to facilitate video packaging and delivery.

PACKAGING GROUPS:

Packaging groups in AWS Elemental MediaPackage serve as organizational units for video content. They enable users to group related assets together and apply consistent settings for packaging and delivery. Each packaging group encapsulates settings for encryption, authorization, and packaging configurations.

PACKAGING CONFIGURATIONS:

Packaging configurations define how video content is formatted and packaged for delivery. They encompass parameters such as the type of manifest (e.g., HLS, DASH), segment duration, ad insertion settings, and

encryption options. Users can create multiple packaging configurations and associate them with different packaging groups to accommodate various content delivery requirements.

ASSETS:

Assets represent the individual video files ingested into AWS Elemental MediaPackage for processing and delivery. Each asset is assigned a unique identifier known as an asset ID and is associated with a specific packaging group. Assets can be ingested from various sources, including Amazon S3 buckets, and are processed according to the specified packaging configurations.

ENDPOINTS:

Endpoints are URLs through which viewers can access the packaged video content for streaming. AWS Elemental MediaPackage generates endpoints for each asset, enabling viewers to stream the content over HTTP or HTTPS protocols. These endpoints are typically associated with CloudFront distributions for efficient content delivery to viewers worldwide.

CLOUDFRONT DISTRIBUTION:

Amazon CloudFront is a globally distributed content delivery network (CDN) that caches and delivers video content to viewers with low latency and high transfer speeds. CloudFront distributions serve as the entry points for accessing the packaged video content, caching the content at edge locations close to viewers for faster delivery.

ORIGIN ENDPOINTS:

Origin endpoints represent the origins from which CloudFront distributions retrieve video content. In the context of AWS Elemental MediaPackage, origin endpoints are associated with packaging groups and provide access to the packaged video content stored within MediaPackage. CloudFront distributions use origin endpoints to fetch the content for delivery to viewers.

CHANNEL:

Channels are a higher-level abstraction in AWS Elemental MediaPackage that group together related assets and endpoints associated with a specific streaming workflow. Channels can be used to organize and manage streaming workflows, apply access control settings, and monitor performance metrics. They provide a convenient way to manage and deliver content to viewers across different devices and platforms.

IAM ROLE:

IAM (Identity and Access Management) Role is a fundamental component of AWS security infrastructure, enabling granular control over access to AWS resources and services. It defines permissions and policies for entities such as AWS Lambda functions, EC2 instances, and users, allowing them to interact securely with AWS services. IAM Roles follow the principle of least privilege, granting only the permissions necessary for specific tasks, reducing the risk of unauthorized access and potential security breaches. IAM Roles are used to delegate permissions across AWS accounts and services, facilitating secure communication and resource sharing. Additionally, IAM Roles support temporary credentials, enabling applications and services to assume roles temporarily for authorized actions, enhancing security and access management in dynamic environments.

HOW IAM ROLE USED HERE:

In this project, IAM (Identity and Access Management) roles are likely used to define and manage permissions for the AWS services and resources involved. Here's how IAM roles might be utilized in different components of this project:

LAMBDA EXECUTION ROLE:

Each Lambda function requires an IAM execution role that grants it permissions to access other AWS services. For example:

- The Lambda function triggered by S3 uploads needs permissions to read from the S3 bucket where videos are uploaded and write to the destination S3 bucket for converted videos.
- The Lambda function triggered by the storage of converted videos may need permissions to interact with DynamoDB to store metadata and to create resources in MediaPackage.

MEDIA CONVERT ROLE:

The IAM role associated with the MediaConvert service will define what actions MediaConvert is allowed to perform. This might include accessing S3 buckets for input and output, logging, and possibly interacting with other AWS services depending on your configuration.

MEDIA PACKAGE ROLE:

The IAM role associated with MediaPackage defines permissions for MediaPackage to perform actions such as ingesting video streams, managing packaging configurations, and delivering content to

viewers. This role might include permissions to access S3 buckets for storing content and to interact with other AWS services like CloudFront for content delivery.

AWS SECRETS MANAGER ROLE:

If you're using AWS Secrets Manager to securely store sensitive information such as API keys, credentials, or configuration data, you would have an IAM role associated with AWS Secrets Manager. This role would grant permissions to read secrets from Secrets Manager for your Lambda functions or other components that need access to these secrets during runtime.

CLOUDFRONT ROLE:

When creating CloudFront distributions, an IAM role is often used to define permissions for CloudFront to access the origin (the S3 bucket containing the videos) and to perform other necessary tasks like logging or managing ACM (AWS Certificate Manager) certificates for HTTPS connections.

DYNAMODB ROLE:

The IAM role associated with DynamoDB will determine the permissions granted to Lambda functions or other services that need to interact with the DynamoDB table. This might include permissions to read and write to the table for storing and retrieving endpoint information.

CLOUDWATCH:

CloudWatch is a comprehensive monitoring and observability service provided by Amazon Web Services (AWS) for cloud resources and applications. It allows users to collect and track metrics, log files, and events from various AWS services and resources in real-time. With CloudWatch, users can gain insights into the performance, health, and operational status of their applications and infrastructure, enabling them to troubleshoot issues, optimize resource utilization, and ensure the overall reliability and availability of their cloud-based systems. Additionally, CloudWatch offers features such as alarms, dashboards, and automated actions to help users proactively monitor and manage their AWS environment efficiently.

HOW CLOUDWATCH USED HERE:

CloudWatch, AWS's monitoring and observability service, can be utilized in several ways within your project:

LAMBDA FUNCTION LOGGING:

CloudWatch Logs can capture the logs generated by your Lambda functions. This includes any print

statements or log messages you include in your Lambda function code. You can use these logs for debugging, monitoring, and troubleshooting your Lambda functions.

MONITORING LAMBDA METRICS:

CloudWatch Metrics provides insights into the performance of your Lambda functions. You can monitor metrics such as invocation count, duration, errors, and throttles. Setting up alarms based on these metrics allows you to be alerted when certain thresholds are exceeded, helping you proactively manage your Lambda functions.

S3 EVENT LOGGING:

If you've configured S3 event notifications to trigger your Lambda functions upon file uploads, you can monitor these events using CloudWatch Events. CloudWatch Events provides a detailed history of events and allows you to set up rules to trigger actions based on these events.

MEDIA CONVERT AND MEDIA PACKAGE LOGGING:

If enabled, MediaConvert and MediaPackage can send logs to CloudWatch Logs. This includes information about video conversion jobs, package configurations, ingestions, and deliveries. By analyzing these logs, you can gain insights into the performance and health of your video processing pipeline.

CLOUDFRONT ACCESS LOGS:

CloudFront access logs can be configured to be stored in an S3 bucket and optionally, streamed to CloudWatch Logs. These logs contain information about requests made to your CloudFront distributions, including details such as the requester's IP address, requested URLs, and response status codes. Analyzing these logs can provide valuable insights into your content delivery performance and help you optimize your distribution configuration.

MONITORING DYNAMODB:

If you're using DynamoDB to store metadata or other information, you can monitor its performance using CloudWatch Metrics. This includes metrics such as read and write capacity utilization, throttled requests, and table activity.

By leveraging CloudWatch, you can gain visibility into the various components of your video processing and delivery pipeline, monitor their performance, and set up alerts to respond to any issues proactively. Integrating CloudWatch effectively into your project can help ensure its reliability, performance, and scalability.

HOW TO MONITOR LOGS FROM LAMBDA FUNCTIONS:

- In the Lambda function console, you'll see a dashboard with various tabs. Click on the "Monitoring" tab.
- In the Monitoring tab, you'll see invocation metrics such as invocation count, duration, and errors. This gives you an overview of the function's performance.
- To view detailed logs, click on the "View logs in CloudWatch" link located below the metrics. This will redirect you to the CloudWatch Logs Insights page.
- In the CloudWatch Logs Insights page, you'll see logs generated by the selected Lambda function. You can filter logs based on various parameters such as time range, log stream, and log group.
- Click on individual log entries to view detailed log messages. You can see information such as timestamp, log message, and log stream.

Note: All Services Must Be Created in the Same Region

Before proceeding with creating an S3 bucket or any other AWS service for your project, it's essential to ensure that all services are created within the same AWS region. This ensures optimal performance and avoids data transfer costs incurred when services are spread across multiple regions. When creating your S3 bucket, be sure to select the same region where your other project components, such as Lambda functions and DynamoDB tables, are located. Consistency in the region selection simplifies network configuration and data transfer between services, contributing to a more streamlined and efficient project setup.

S3 BUCKET:

An S3 bucket serves as the core storage unit within Amazon Simple Storage Service (S3), a widely-used object storage solution provided by AWS. Acting as a container for diverse data objects like files, images, videos, and backups, each bucket possesses a globally unique name across all AWS accounts and regions. This universal namespace ensures the uniqueness of bucket identifiers. S3 buckets offer exceptional scalability, effortlessly accommodating vast amounts of data while AWS manages underlying storage infrastructure scaling seamlessly. Data stored within S3 buckets is easily accessible over the internet via HTTP or HTTPS using unique URLs generated by AWS. Access to stored objects can be finely controlled through various mechanisms such as bucket policies, Access Control Lists (ACLs), and IAM policies. Additionally, S3 guarantees high durability and availability by redundantly storing data across multiple devices and facilities within the selected AWS region.

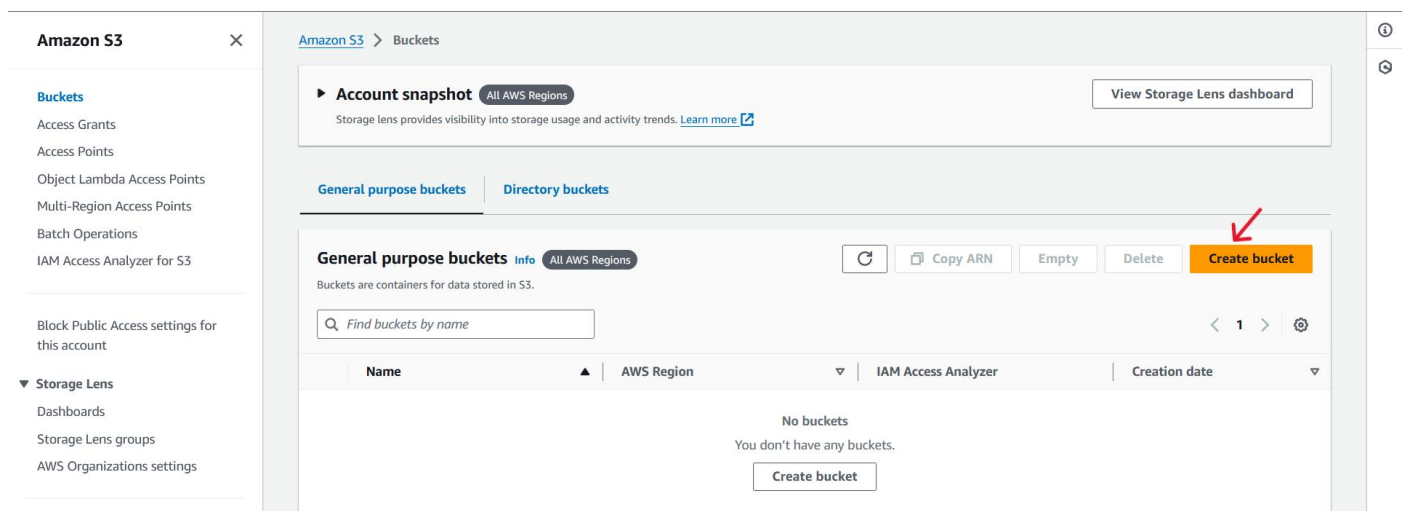
Advanced features like lifecycle policies facilitate automated management of objects, allowing for transitions between different storage classes or scheduled deletions. With support for versioning, security

features such as encryption at rest and in transit, and comprehensive audit logging, S3 buckets provide a secure, reliable, and flexible solution for storing and managing data in the cloud.

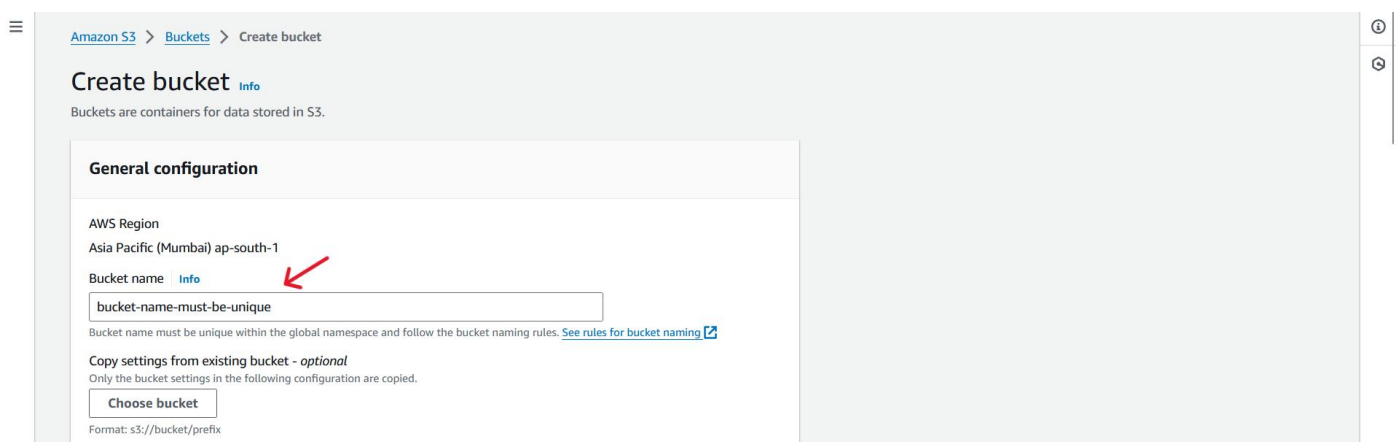
CREATE AN BUCKET TO STORE THE INPUT FILES IN S3:

To create an S3 bucket in your project, follow these steps:

- Navigate to the AWS Management Console at <https://aws.amazon.com/console/> and sign in to your AWS account.
- Once signed in, search for "S3" in the AWS Management Console search bar or navigate to the "Storage" section and select "S3".
- In the S3 dashboard, click on the "Create bucket" button to initiate the bucket creation process.

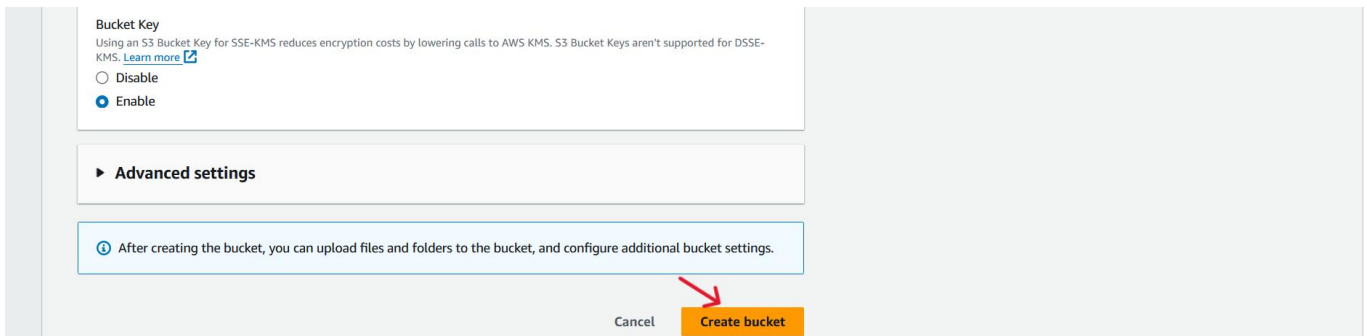


- Enter a unique name for your bucket. This name must be globally unique across all AWS accounts and regions.



- Choose the AWS region where you want to create the bucket. Consider selecting a region that is geographically closest to your intended users to minimize latency.

- Optionally, configure additional settings such as versioning, server access logging, encryption, and tags based on your project requirements.
- Define the access permissions for your bucket. By default, buckets are private, meaning that only the bucket owner has access to the contents. You can configure bucket policies and access control lists (ACLs) to grant access to specific IAM users, roles, or even make the bucket publicly accessible if needed.
- Review the bucket configuration settings to ensure they align with your requirements.
- Click on the "Create bucket" button to finalize the creation of your S3 bucket.



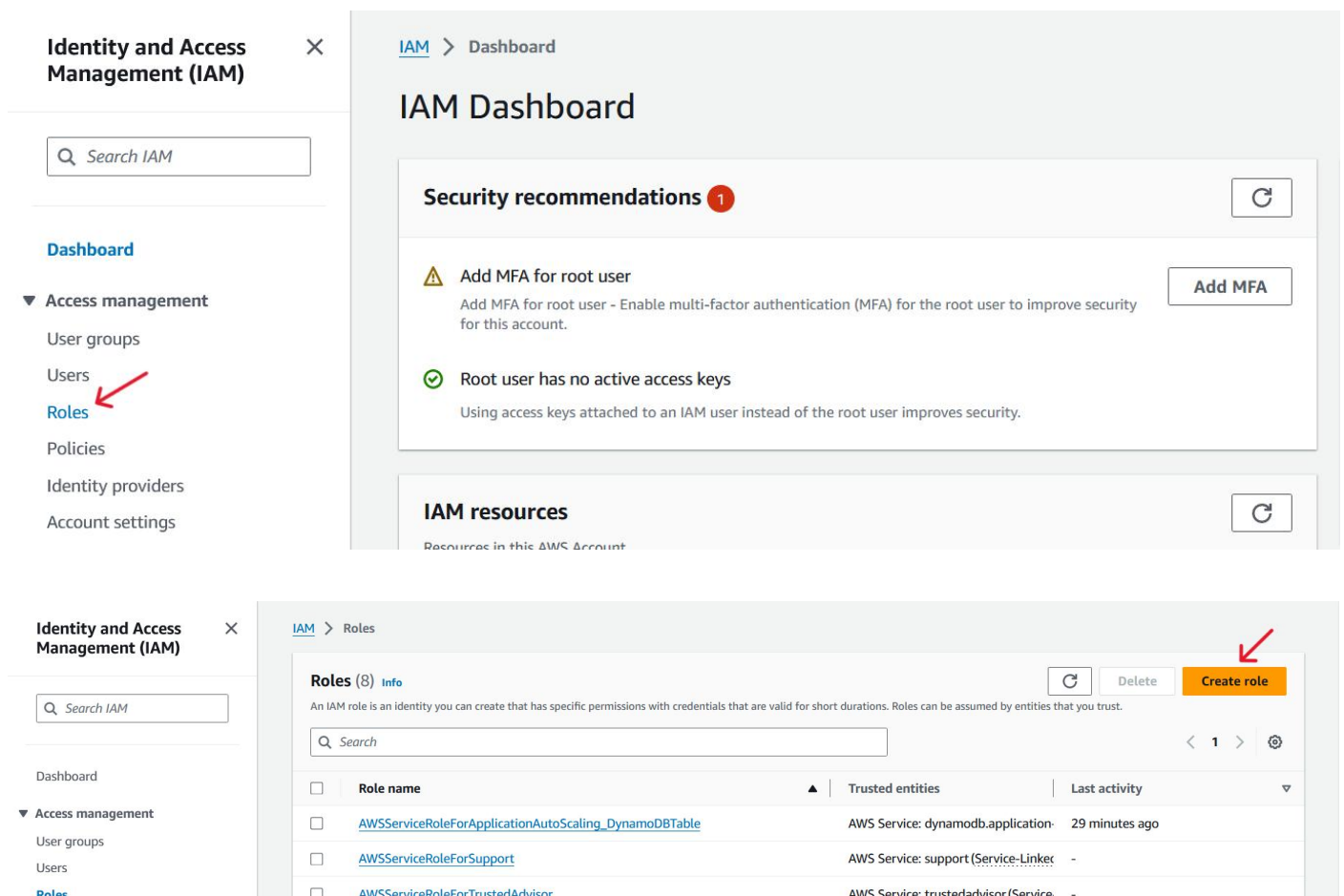
- Once the bucket is created, you can access it from the S3 dashboard to upload, download, and manage objects.
- Obtain the bucket's unique URL, which follows the format `https://<bucket-name>.s3.amazonaws.com/`, for accessing objects programmatically or sharing with others.
- Update your project configurations or code to interact with the newly created S3 bucket as needed. For example, you may configure Lambda functions to trigger upon S3 events or specify the bucket as a destination for uploaded files.
- As well, you create **another bucket** to store the **output files** for separate organization, like an input-bucket for one and an output-bucket for another. Note that the bucket name should be unique globally.

LAMBDA:

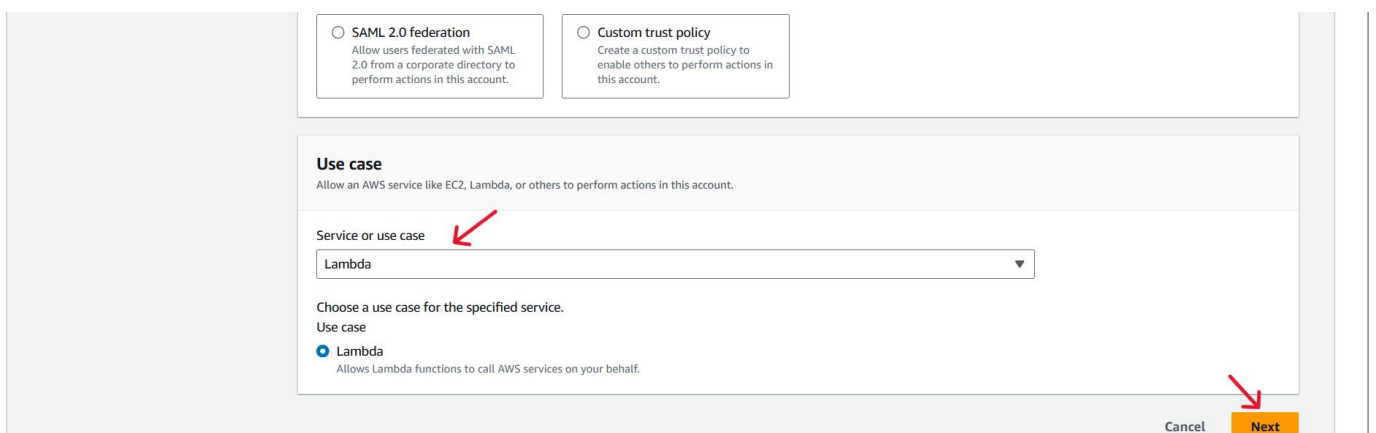
Lambda, in the context of computing, refers to AWS Lambda, a serverless compute service provided by Amazon Web Services (AWS). It enables users to run code without provisioning or managing servers, allowing for seamless scaling and cost optimization. With Lambda, developers can upload their code, which is then automatically executed in response to predefined events, such as HTTP requests, file uploads, or database updates. Lambda functions can be written in various programming languages, including Python, Node.js, Java, and more, providing flexibility for developers. Key benefits of Lambda include its event-driven architecture, pay-per-use pricing model, and automatic scaling, making it ideal for building highly scalable and cost-effective applications and services in the cloud.

CREATE AN IAM ROLE FOR LAMBDA FUNCTION:

- Open the AWS Management Console and go to the IAM service.
- In the IAM dashboard, click on "Roles" in the left sidebar, then click on the "Create role" button.



- Choose "AWS service" as the trusted entity, and select "Lambda" as the service that will use this role.



- Click "Next: Permissions" to proceed to the permissions configuration.

- In the "Attach permissions policies" step, search for and select the policies that grant the necessary permissions. For this scenario, you'll need to attach the following policies:

- AmazonS3FullAccess:** Grants full access to Amazon S3 buckets.
- AWSElementalMediaConvertFullAccess:** Provides full access to AWS Elemental MediaConvert for video conversion.
- AWSLambdaBasicExecutionRole:** Basic permissions for Lambda function execution.

Step 1
[Select trusted entity](#)

Step 2
Add permissions

Step 3
Name, review, and create

Add permissions [Info](#)

Permissions policies (3/925) [Info](#)

Choose one or more policies to attach to your new role.

Filter by Type: All types 3 matches

<input type="checkbox"/>	Policy name	Type	Description
<input checked="" type="checkbox"/>	AWSElementalMediaConvertFullAccess	AWS managed	Provides full access to AWS Elemental ...
<input type="checkbox"/>	AWSElementalMediaConvertReadOnly	AWS managed	Provides read only access to AWS Elem...
<input type="checkbox"/>	MediaConnectGatewayInstanceRolePolicy	AWS managed	This policy grants permission to regist...

► Set permissions boundary - optional

Cancel Previous **Next**

- Review the policies attached to the role to ensure they grant the necessary permissions.
- Give the role a descriptive name that reflects its purpose, such as "VODLambdaRole" for the Role name.

IAM > Roles > Create role

Step 1
[Select trusted entity](#)

Step 2
[Add permissions](#)

Step 3
Name, review, and create

Name, review, and create

Role details

Role name
Enter a meaningful name to identify this role.
VODLambdaRole
Maximum 64 characters. Use alphanumeric and '+,=,@,_,-' characters.

Description

- Choose Create role.

[AmazonS3FullAccess](#) AWS managed Permissions policy

[AWSElementalMediaConvertFullAccess](#) AWS managed Permissions policy

[AWSLambdaBasicExecutionRole](#) AWS managed Permissions policy

Step 3: Add tags

Add tags - optional [Info](#)

Tags are key-value pairs that you can add to AWS resources to help identify, organize, or search for resources.

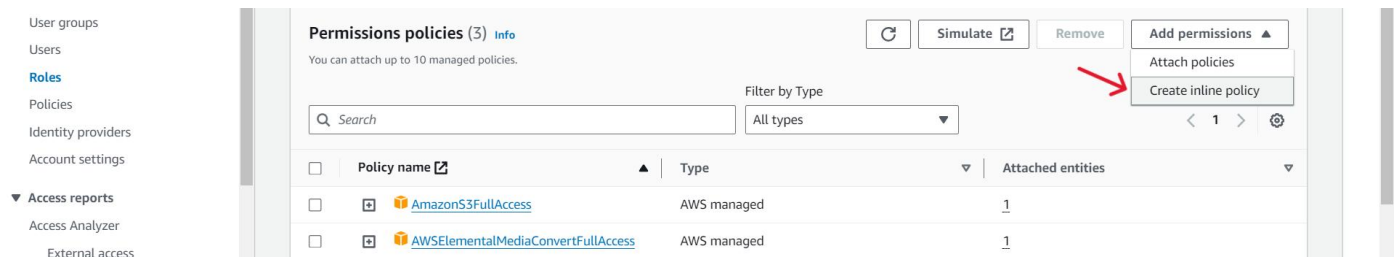
No tags associated with the resource.

[Add new tag](#)

You can add up to 50 more tags.

Cancel Previous **Create role**

- Type VODLambdaRole into the filter box on the Roles page and choose the role you just created.
- On the Permissions tab, click on the Add Inline Policy link and choose the JSON tab.



- Copy and paste the following JSON in the Policy Document Box. You will need to edit this policy in the next step to fill in the resources for your application. Click "Create role" to finalize the role creation process.

```

1  {
2    "Version": "2012-10-17",
3    "Statement": [
4      {
5        "Action": [
6          "logs:CreateLogGroup",
7          "logs:CreateLogStream",
8          "logs:PutLogEvents"
9        ],
10       "Resource": "*",
11       "Effect": "Allow",
12       "Sid": "Logging"
13     },
14     {
15       "Action": [
16         "iam:PassRole"
17       ],
18       "Resource": [
19         "<ARN for vod-MediaConvertRole>"
20       ],
21       "Effect": "Allow",
22       "Sid": "PassRole"
23     },
24     {
25       "Action": [
26         "mediaconvert:*"
27       ],
28       "Resource": [
29         "*"
30       ],
31       "Effect": "Allow",
32       "Sid": "MediaConvertService"
33     }
34   ]
35 }

```

- Replace tag in the policy with the ARN for the vod-MediaConvertRole you created earlier.
- Once reviewed , click on the Next button.

- Enter VODLambdaPolicy in the Policy Name box.
- Click on the Create Policy button.

Policy details

Policy name
Enter a meaningful name to identify this policy.

VODLambdaPolicy

Maximum 128 characters. Use alphanumeric and "+-=, @ _ . /" characters.

Permissions defined in this policy

Permissions defined in this policy document specify which actions are allowed or denied. To define permissions for an IAM identity (user, user group, or role), attach a policy to it.

Q Search

Allow (3 of 408 services) Show remaining 405 services

Service	Access level	Resource	Request condition
CloudWatch Logs	Limited: Write	All resources	None
IAM	Limited: Write	RoleName string like [MediaConvertRole	None
MediaConvert	Full access	All resources	None

Cancel Previous **Create policy**

UPDATE LAMBDA FUNCTION:

- After creating the IAM role, you need to associate it with your Lambda function.
- Go to the Lambda service in the AWS Management Console and select your Lambda function.
- Under the "Configuration" tab, scroll down to the "Permissions" section.
- Click on "Edit" in the "Execution role" row, and select "Use an existing role".
- Choose the IAM role you created earlier from the dropdown list.
- Save the changes to associate the IAM role with your Lambda function.

CREATE AN IAM ROLE FOR MEDIA CONVERT:

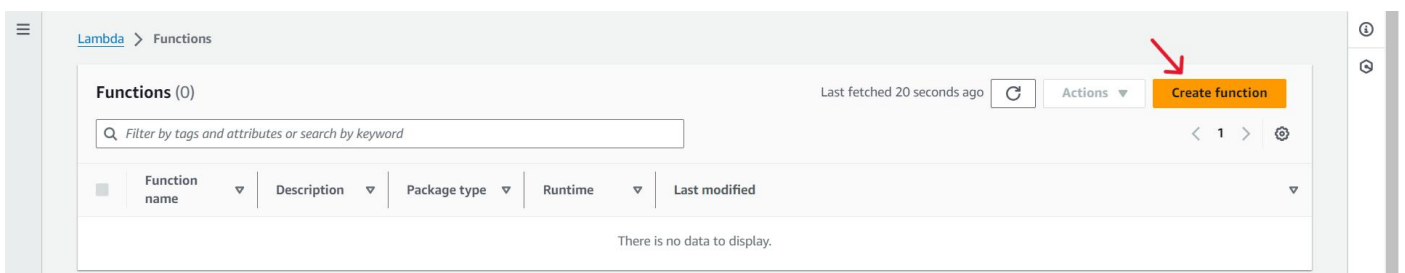
- Open the AWS Management Console and go to the IAM service.
- In the IAM dashboard, click on "Roles" in the left sidebar, then click on the "Create role" button.
- Choose "AWS service" as the trusted entity, and select "Lambda" as the service that will use this role.
- Click "Next: Permissions" to proceed to the permissions configuration.
- In the "Attach permissions policies" step, search for and select the policies that grant the necessary permissions. For this scenario, you'll need to attach the following policies:

AmazonS3FullAccess: Grants full access to Amazon S3 buckets.

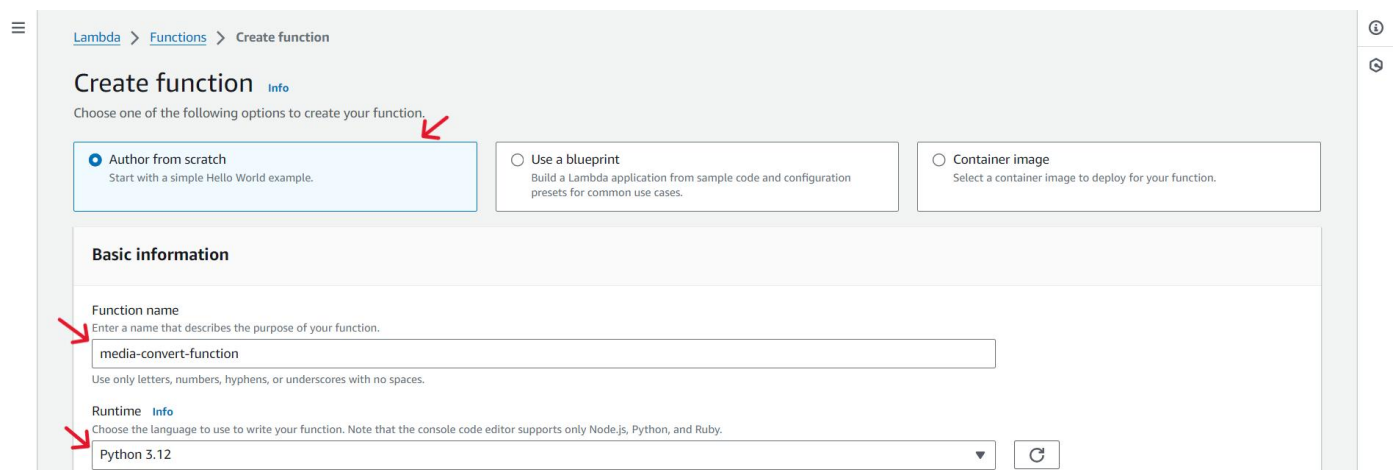
- Review the policies attached to the role to ensure they grant the necessary permissions.
- Give the role a descriptive name that reflects its purpose, such as “MediaConvertRole” for the Role name.
- Choose Create role.

CREATE A FUNCTION IN AWS LAMBDA FOR MEDIA CONVERT:

- Sign in to your AWS account at <https://aws.amazon.com/console/>.
- Navigate to the Lambda service by searching for "Lambda" in the AWS Management Console.
- In the Lambda console, click on the "Create function" button.



- Choose the "Author from scratch" option to create a new function from scratch.
- Configure Basic Information:
- Enter a name for your function in the "Function name" field.
- Select the runtime environment for your function (e.g., Python, Node.js, Java).



- Choose an existing execution role or create a new one with necessary permissions.
- Once you've configured the basic information, click on the "Create function" button to create your Lambda function.

Architecture [Info](#)
Choose the instruction set architecture you want for your function code.

☒ x86_64
☐ arm64

Permissions [Info](#)
By default, Lambda will create an execution role with permissions to upload logs to Amazon CloudWatch Logs. You can customize this default role later when adding triggers.

▼ **Change default execution role**

Execution role
Choose a role that defines the permissions of your function. To create a custom role, go to the [IAM console](#).

☐ Create a new role with basic Lambda permissions
☒ Use an existing role
☐ Create a new role from AWS policy templates

Existing role
Choose an existing role that you've created to be used with this Lambda function. The role must have permission to upload logs to Amazon CloudWatch Logs.

VODLambdaRole

[View the VODLambdaRole role](#) on the IAM console.

► **Advanced settings**

[Cancel](#) [Create function](#)

- In the function code editor, you see the sample return type program.
- In the Lambda console, locate the "Test" button on the top right corner of the code editor.
- Click on the dropdown arrow next to the "Test" button and select "Configure test events."
- Choose "Create new test event" and enter a name for your test event.
- Or pre- build choose a template based on your preference.
- Define the test event payload in JSON format based on the expected input to your Lambda function.
- Include any required parameters or data that your function expects as input.
- For example, if your Lambda function expects a file upload event, you might include metadata about the uploaded file in the test event payload.
- Once you've defined the test event payload, click on the "Create" button to save the test event.
- After creating the test event, Click on the "Test" button to execute your Lambda function with the test event payload.
- After running the test, review the output of your Lambda function in the "Execution result" section.
- Verify that the function behaves as expected and produces the desired output based on the test event.
- Additionally, you can view any logs or error messages generated by the function execution in the "Logs" tab.

LAMBDA INTEGRATION WITH S3 BUCKET:

To integrate Lambda with S3 so that Lambda triggers automatically when an object is uploaded:

- Create a Lambda function that contains the code you want to execute when a new object is uploaded to S3.
- In the AWS Management Console, navigate to Lambda service, select your Lambda function, and click on "Add trigger." Choose S3 as the trigger type.

▼ Function overview [Info](#)

Diagram

Template



media-convert-function



Layers

(0)

+ Add trigger

+ Add destination

- Specify the S3 bucket and the event type (e.g., ObjectCreated) that should trigger the Lambda function.

Trigger configuration [Info](#)



aws asynchronous storage

Bucket

Choose or enter the ARN of an S3 bucket that serves as the event source. The bucket must be in the same region as the function.

s3/mc-input-buckets

Bucket region: ap-south-1

Event types

Select the events that you want to have trigger the Lambda function. You can optionally set up a prefix or suffix for an event. However, for each bucket, individual events cannot have multiple configurations with overlapping prefixes or suffixes that could match the same object key.

All object create events

PUT

POST

COPY

Multipart upload completed

- Ensure that the Lambda function has the necessary permissions to access the S3 bucket. This typically involves creating an IAM role with permissions to read from the S3 bucket and attaching it to the Lambda function.
- Then, Click the Add the button to add the trigger in the lambda function.

Suffix - optional

Enter a single optional suffix to limit the notifications to objects with keys that end with matching characters.

e.g. .jpg

Recursive invocation

If your function writes objects to an S3 bucket, ensure that you are using different S3 buckets for input and output. Writing to the same bucket increases the risk of creating a recursive invocation, which can result in increased Lambda usage and increased costs. [Learn more](#)

☒ I acknowledge that using the same S3 bucket for both input and output is not recommended and that this configuration can cause recursive invocations, increased Lambda usage, and increased costs.

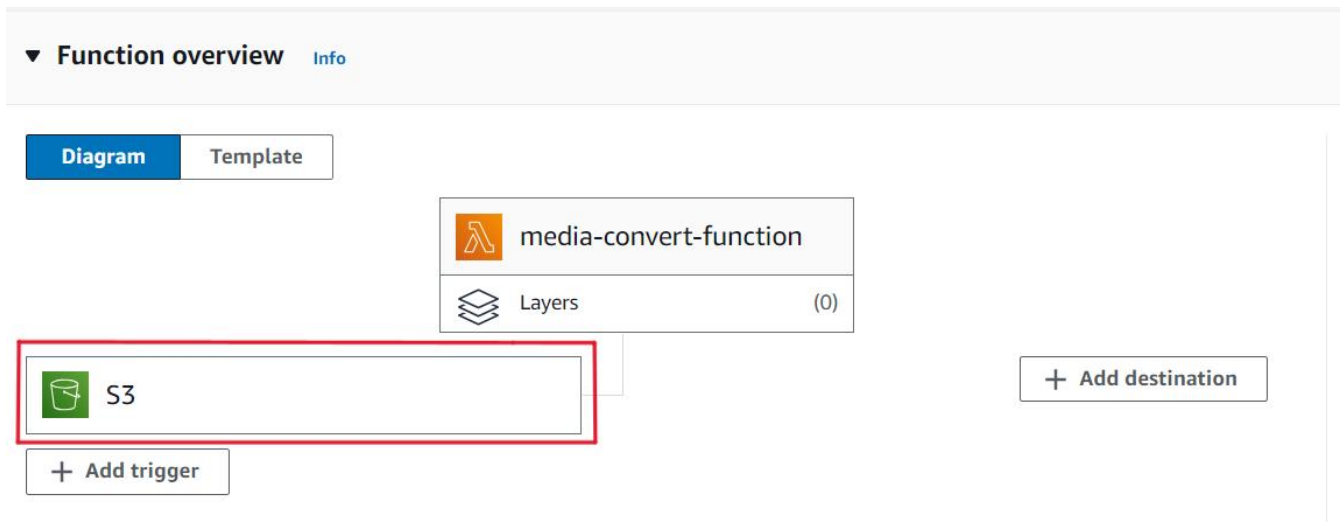
Lambda will add the necessary permissions for AWS S3 to invoke your Lambda function from this trigger. [Learn more](#) about the Lambda permissions model.

Cancel

Add

- Upload an object to the specified S3 bucket and verify that the Lambda function is triggered automatically.
- Monitor the logs in CloudWatch for both the Lambda function and the S3 bucket to troubleshoot any issues and ensure everything is working as expected.

- Ensure that the lambda function has successfully integrated with s3 or not.



CONFIGURE THE LAMBDA CODE EDITOR TO EXECUTE MEDIA CONVERT:

- After lambda integration with S3. You can start to write the code to execute media convert.
- For your reference you can replace the following code instead of existing code in lambda function.

lambda_function.py

```
import glob
import json
import os
import boto3
import datetime
import random

from botocore.client import ClientError

def handler(event, context):

    sourceS3Bucket = event['Records'][0]['s3']['bucket']['name']
    sourceS3Key = event['Records'][0]['s3']['object']['key']
    sourceS3 = 's3://' + sourceS3Bucket + '/' + sourceS3Key
    sourceS3Basename = os.path.splitext(os.path.basename(sourceS3))[0]
    destinationS3 = 's3://' + "Your-Destination-Bucket"
    destinationS3basename = os.path.splitext(os.path.basename(destinationS3))[0]
    mediaConvertRole = "Your Arn of MediaConvert Role"
    region = "Your-Default-Region"
    statusCode = 200
    body = {}

    print (json.dumps(event))

    try:
        # Job settings are in the current working directory
```

```

with open('job.json') as json_data:
    jobSettings = json.load(json_data)
    print(jobSettings)

# get the account-specific mediaconvert endpoint for this region
mc_client = boto3.client('mediaconvert', region_name=region)
endpoints = mc_client.describe_endpoints()

# add the account-specific endpoint to the client session
client = boto3.client('mediaconvert', region_name=region, endpoint_url=endpoints['Endpoints'][0]['Url'],
verify=False)

# Update the job settings with the source video from the S3 event and destination
# paths for converted videos
jobSettings['Inputs'][0]['FileInput'] = sourceS3

PackagingGroup = 'youtube-group'

S3KeyHLS = PackagingGroup + '/assets/' + '/HLS/' + sourceS3Basename
jobSettings['OutputGroups'][0]['OutputGroupSettings']['HlsGroupSettings']['Destination'] \
    = destinationS3 + '/' + S3KeyHLS

S3KeyWatermark = PackagingGroup + '/assets/' + '/MP4/' + sourceS3Basename
jobSettings['OutputGroups'][1]['OutputGroupSettings']['FileGroupSettings']['Destination'] \
    = destinationS3 + '/' + S3KeyWatermark

S3KeyThumbnails = PackagingGroup + '/assets/' + '/Thumbnails/' + sourceS3Basename
jobSettings['OutputGroups'][2]['OutputGroupSettings']['FileGroupSettings']['Destination'] \
    = destinationS3 + '/' + S3KeyThumbnails

S3KeyDASH = PackagingGroup + '/assets/' + '/DASH/' + sourceS3Basename
jobSettings['OutputGroups'][3]['OutputGroupSettings']['DashIsoGroupSettings']['Destination'] \
    = destinationS3 + '/' + S3KeyDASH

print('jobSettings:')
print(json.dumps(jobSettings))

# Convert the video using AWS Elemental MediaConvert
job = client.create_job(Role=mediaConvertRole, UserMetadata=jobMetadata, Settings=jobSettings)
print (json.dumps(job, default=str))

print(event)

except Exception as e:
    print ('Exception: %s' % e)
    statusCode = 500
    raise

finally:
    return {
        'statusCode': statusCode,
        'body': json.dumps(body),

```

```

    'headers': {'Content-Type': 'application/json', 'Access-Control-Allow-Origin': '*'}
  }
}

```

- Replace your own destination bucket, ARN of mediaconvert role and default region.
- Create a new file in the current working directory and name as job.json
- In the json file paste the following code to create a job in mediaconvert.

job.json

```

{
  "OutputGroups": [
    {
      "CustomName": "HLS",
      "Name": "Apple HLS",
      "Outputs": [
        {
          "ContainerSettings": {
            "Container": "M3U8",
            "M3u8Settings": {
              "AudioFramesPerPes": 4,
              "PcrControl": "PCR_EVERY_PES_PACKET",
              "PmtPid": 480,
              "PrivateMetadataPid": 503,
              "ProgramNumber": 1,
              "PatInterval": 0,
              "PmtInterval": 0,
              "Scte35Source": "NONE",
              "TimedMetadata": "NONE",
              "VideoPid": 481,
              "AudioPids": [
                482,
                483,
                484,
                485,
                486,
                487,
                488,
                489,
                490,
                491,
                492
              ]
            }
          },
          "VideoDescription": {
            "Width": 640,
            "ScalingBehavior": "DEFAULT",
            "Height": 360,
            "TimecodeInsertion": "DISABLED",
            "AntiAlias": "ENABLED",
            "Sharpness": 50,

```



```

"CodecSettings": {
  "Codec": "H_264",
  "H264Settings": {
    "InterlaceMode": "PROGRESSIVE",
    "NumberReferenceFrames": 3,
    "Syntax": "DEFAULT",
    "Softness": 0,
    "GopClosedCadence": 1,
    "GopSize": 90,
    "Slices": 1,
    "GopBReference": "DISABLED",
    "SlowPal": "DISABLED",
    "SpatialAdaptiveQuantization": "ENABLED",
    "TemporalAdaptiveQuantization": "ENABLED",
    "FlickerAdaptiveQuantization": "DISABLED",
    "EntropyEncoding": "CABAC",
    "FramerateControl": "INITIALIZE_FROM_SOURCE",
    "RateControlMode": "QVBR",
    "QvbrSettings": {
      "QvbrQualityLevel": 7
    },
    "MaxBitrate": 1000000,
    "CodecProfile": "MAIN",
    "Telecine": "NONE",
    "MinIInterval": 0,
    "AdaptiveQuantization": "HIGH",
    "CodecLevel": "AUTO",
    "FieldEncoding": "PAFF",
    "SceneChangeDetect": "ENABLED",
    "QualityTuningLevel": "SINGLE_PASS",
    "FramerateConversionAlgorithm": "DUPLICATE_DROP",
    "UnregisteredSeiTimecode": "DISABLED",
    "GopSizeUnits": "FRAMES",
    "ParControl": "INITIALIZE_FROM_SOURCE",
    "NumberBFramesBetweenReferenceFrames": 2,
    "RepeatPps": "DISABLED"
  }
},
"AfdSignaling": "NONE",
"DropFrameTimecode": "ENABLED",
"RespondToAfd": "NONE",
"ColorMetadata": "INSERT"
},
"AudioDescriptions": [
{
  "AudioTypeControl": "FOLLOW_INPUT",
  "CodecSettings": {
    "Codec": "AAC",
    "AacSettings": {
      "AudioDescriptionBroadcasterMix": "NORMAL",
      "Bitrate": 96000,
      "RateControlMode": "CBR",
      "CodecProfile": "LC",
      "CodingMode": "CODING_MODE_2_0",

```

```

        "RawFormat": "NONE",
        "SampleRate": 48000,
        "Specification": "MPEG4"
    }
},
    "LanguageCodeControl": "FOLLOW_INPUT"
}
],
"OutputSettings": {
    "HlsSettings": {
        "AudioGroupId": "program_audio",
        "AudioRenditionSets": "program_audio",
        "SegmentModifier": "$dt$",
        "IFrameOnlyManifest": "EXCLUDE"
    }
},
    "NameModifier": "_360"
},
{
    "ContainerSettings": {
        "Container": "M3U8",
        "M3u8Settings": {
            "AudioFramesPerPes": 4,
            "PcrControl": "PCR_EVERY_PES_PACKET",
            "PmtPid": 480,
            "PrivateMetadataPid": 503,
            "ProgramNumber": 1,
            "PatInterval": 0,
            "PmtInterval": 0,
            "Scte35Source": "NONE",
            "Scte35Pid": 500,
            "TimedMetadata": "NONE",
            "TimedMetadataPid": 502,
            "VideoPid": 481,
            "AudioPids": [
                482,
                483,
                484,
                485,
                486,
                487,
                488,
                489,
                490,
                491,
                492
            ]
        }
    }
},
"VideoDescription": {
    "Width": 960,
    "ScalingBehavior": "DEFAULT",
    "Height": 540,
    "TimecodeInsertion": "DISABLED",

```

```

"AntiAlias": "ENABLED",
"Sharpness": 50,
"CodecSettings": {
  "Codec": "H_264",
  "H264Settings": {
    "InterlaceMode": "PROGRESSIVE",
    "NumberReferenceFrames": 3,
    "Syntax": "DEFAULT",
    "Softness": 0,
    "GopClosedCadence": 1,
    "GopSize": 90,
    "Slices": 1,
    "GopBReference": "DISABLED",
    "SlowPal": "DISABLED",
    "SpatialAdaptiveQuantization": "ENABLED",
    "TemporalAdaptiveQuantization": "ENABLED",
    "FlickerAdaptiveQuantization": "DISABLED",
    "EntropyEncoding": "CABAC",
    "MaxBitrate": 2000000,
    "FramerateControl": "INITIALIZE_FROM_SOURCE",
    "RateControlMode": "QVBR",
    "QvbrSettings": {
      "QvbrQualityLevel": 7
    },
    "CodecProfile": "MAIN",
    "Telecine": "NONE",
    "MinIInterval": 0,
    "AdaptiveQuantization": "HIGH",
    "CodecLevel": "AUTO",
    "FieldEncoding": "PAFF",
    "SceneChangeDetect": "ENABLED",
    "QualityTuningLevel": "SINGLE_PASS",
    "FramerateConversionAlgorithm": "DUPLICATE_DROP",
    "UnregisteredSeiTimecode": "DISABLED",
    "GopSizeUnits": "FRAMES",
    "ParControl": "INITIALIZE_FROM_SOURCE",
    "NumberBFramesBetweenReferenceFrames": 2,
    "RepeatPps": "DISABLED"
  },
  "AfdSignaling": "NONE",
  "DropFrameTimecode": "ENABLED",
  "RespondToAfd": "NONE",
  "ColorMetadata": "INSERT"
},
"AudioDescriptions": [
  {
    "AudioTypeControl": "FOLLOW_INPUT",
    "CodecSettings": {
      "Codec": "AAC",
      "AacSettings": {
        "AudioDescriptionBroadcasterMix": "NORMAL",
        "Bitrate": 96000,
        "RateControlMode": "CBR",

```

```

        "CodecProfile": "LC",
        "CodingMode": "CODING_MODE_2_0",
        "RawFormat": "NONE",
        "SampleRate": 48000,
        "Specification": "MPEG4"
    }
},
    "LanguageCodeControl": "FOLLOW_INPUT"
}
],
"OutputSettings": {
    "HlsSettings": {
        "AudioGroupId": "program_audio",
        "AudioRenditionSets": "program_audio",
        "SegmentModifier": "$dt$",
        "IFrameOnlyManifest": "EXCLUDE"
    }
},
    "NameModifier": "_540"
},
{
    "ContainerSettings": {
        "Container": "M3U8",
        "M3u8Settings": {
            "AudioFramesPerPes": 4,
            "PcrControl": "PCR_EVERY_PES_PACKET",
            "PmtPid": 480,
            "PrivateMetadataPid": 503,
            "ProgramNumber": 1,
            "PatInterval": 0,
            "PmtInterval": 0,
            "Scte35Source": "NONE",
            "Scte35Pid": 500,
            "TimedMetadata": "NONE",
            "TimedMetadataPid": 502,
            "VideoPid": 481,
            "AudioPids": [
                482,
                483,
                484,
                485,
                486,
                487,
                488,
                489,
                490,
                491,
                492
            ]
        }
    }
},
"VideoDescription": {
    "Width": 1280,
    "ScalingBehavior": "DEFAULT",

```

```

    "Height": 720,
    "TimecodeInsertion": "DISABLED",
    "AntiAlias": "ENABLED",
    "Sharpness": 50,
    "CodecSettings": {
        "Codec": "H_264",
        "H264Settings": {
            "InterlaceMode": "PROGRESSIVE",
            "NumberReferenceFrames": 3,
            "Syntax": "DEFAULT",
            "Softness": 0,
            "GopClosedCadence": 1,
            "GopSize": 90,
            "Slices": 1,
            "GopBReference": "DISABLED",
            "SlowPal": "DISABLED",
            "SpatialAdaptiveQuantization": "ENABLED",
            "TemporalAdaptiveQuantization": "ENABLED",
            "FlickerAdaptiveQuantization": "DISABLED",
            "EntropyEncoding": "CABAC",
            "MaxBitrate": 3000000,
            "FramerateControl": "INITIALIZE_FROM_SOURCE",
            "RateControlMode": "QVBR",
            "QvbrSettings": {
                "QvbrQualityLevel": 7
            },
            "CodecProfile": "MAIN",
            "Telecine": "NONE",
            "MinIInterval": 0,
            "AdaptiveQuantization": "HIGH",
            "CodecLevel": "AUTO",
            "FieldEncoding": "PAFF",
            "SceneChangeDetect": "ENABLED",
            "QualityTuningLevel": "SINGLE_PASS",
            "FramerateConversionAlgorithm": "DUPLICATE_DROP",
            "UnregisteredSeiTimecode": "DISABLED",
            "GopSizeUnits": "FRAMES",
            "ParControl": "INITIALIZE_FROM_SOURCE",
            "NumberBFramesBetweenReferenceFrames": 2,
            "RepeatPps": "DISABLED"
        },
        "CodecProfile": "MAIN",
        "Telecine": "NONE",
        "MinIInterval": 0,
        "AdaptiveQuantization": "HIGH",
        "CodecLevel": "AUTO",
        "FieldEncoding": "PAFF",
        "SceneChangeDetect": "ENABLED",
        "QualityTuningLevel": "SINGLE_PASS",
        "FramerateConversionAlgorithm": "DUPLICATE_DROP",
        "UnregisteredSeiTimecode": "DISABLED",
        "GopSizeUnits": "FRAMES",
        "ParControl": "INITIALIZE_FROM_SOURCE",
        "NumberBFramesBetweenReferenceFrames": 2,
        "RepeatPps": "DISABLED"
    },
    "AfdSignaling": "NONE",
    "DropFrameTimecode": "ENABLED",
    "RespondToAfd": "NONE",
    "ColorMetadata": "INSERT"
},
"AudioDescriptions": [
    {
        "AudioTypeControl": "FOLLOW_INPUT",
        "CodecSettings": {
            "Codec": "AAC",
            "AacSettings": {
                "AudioDescriptionBroadcasterMix": "NORMAL",

```

```

        "Bitrate": 96000,
        "RateControlMode": "CBR",
        "CodecProfile": "LC",
        "CodingMode": "CODING_MODE_2_0",
        "RawFormat": "NONE",
        "SampleRate": 48000,
        "Specification": "MPEG4"
    }
},
    "LanguageCodeControl": "FOLLOW_INPUT"
}
],
"OutputSettings": {
    "HlsSettings": {
        "AudioGroupId": "program_audio",
        "AudioRenditionSets": "program_audio",
        "SegmentModifier": "$dt$",
        "IFrameOnlyManifest": "EXCLUDE"
    }
},
    "NameModifier": "_720"
}
],
"OutputGroupSettings": {
    "Type": "HLS_GROUP_SETTINGS",
    "HlsGroupSettings": {
        "ManifestDurationFormat": "INTEGER",
        "SegmentLength": 10,
        "TimedMetadataId3Period": 10,
        "CaptionLanguageSetting": "OMIT",
        "Destination": "s3://output-bucket/assets/HLS/",
        "TimedMetadataId3Frame": "PRIV",
        "CodecSpecification": "RFC_4281",
        "OutputSelection": "MANIFESTS_AND_SEGMENTS",
        "ProgramDateTimePeriod": 600,
        "MinSegmentLength": 0,
        "DirectoryStructure": "SINGLE_DIRECTORY",
        "ProgramDateTime": "EXCLUDE",
        "SegmentControl": "SEGMENTED_FILES",
        "ManifestCompression": "NONE",
        "ClientCache": "ENABLED",
        "StreamInfResolution": "INCLUDE"
    }
}
}
],
"AdAvailOffset": 0,
"Inputs": [
{
    "AudioSelectors": {
        "Audio Selector 1": {
            "Offset": 0,
            "DefaultSelection": "DEFAULT",
            "ProgramSelection": 1
        }
    }
}
]

```

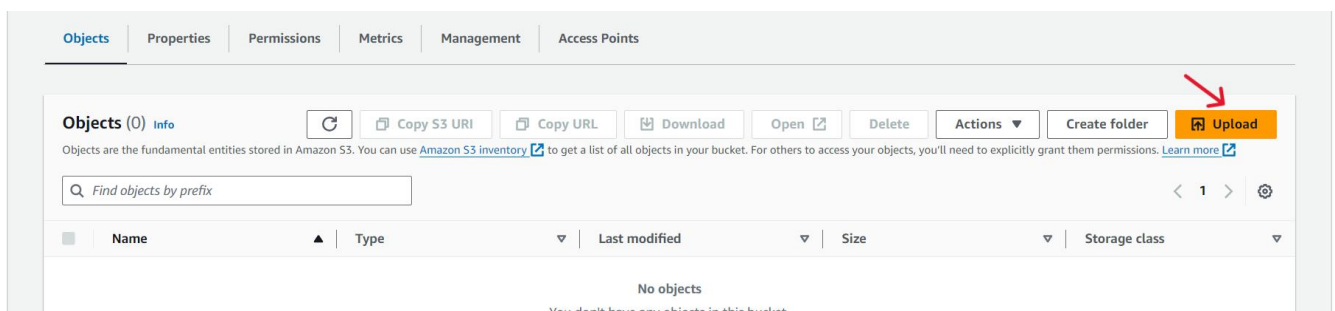
```

    }
  },
  "VideoSelector": {
    "ColorSpace": "FOLLOW"
  },
  "FilterEnable": "AUTO",
  "PsiControl": "USE_PSI",
  "FilterStrength": 0,
  "DeblockFilter": "DISABLED",
  "DenoiseFilter": "DISABLED",
  "TimecodeSource": "EMBEDDED",
  "FileInput": "s3://input-bucket"
}
]
}

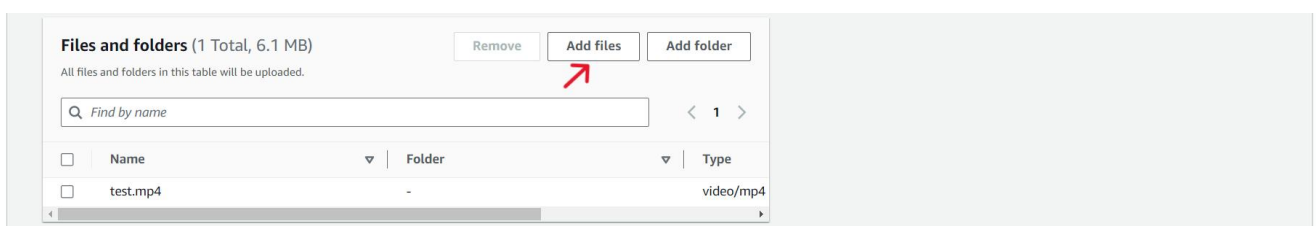
```

TEST THE LAMBDA FUNCTION:

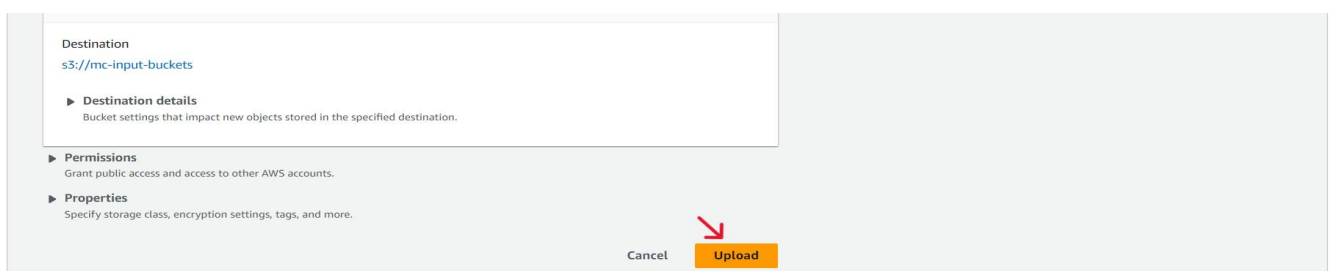
- Go to the S3 bucket you created earlier for input.
- Then, click on upload button.



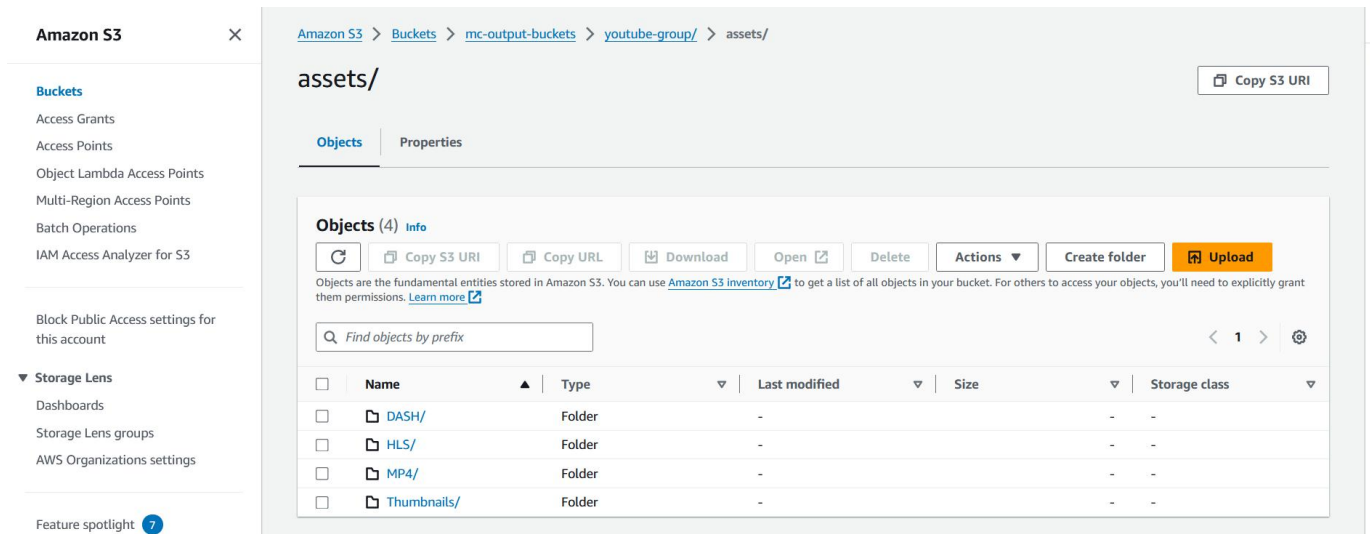
- Click on the "Add file" button.



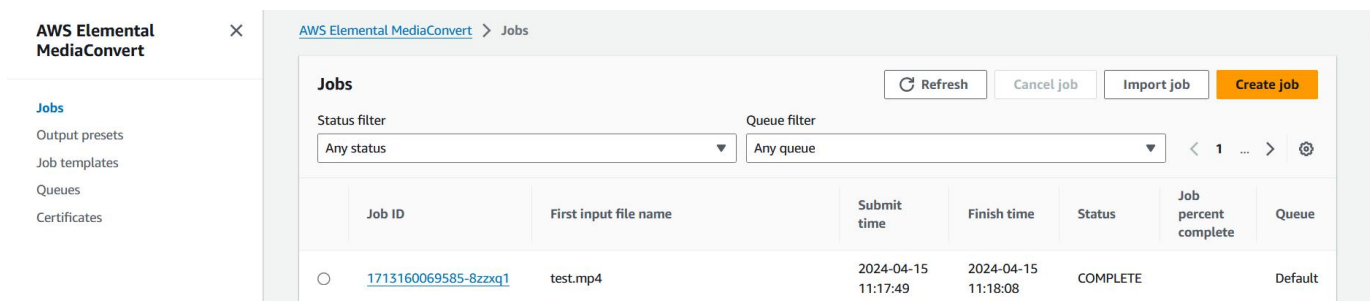
- Select the video file you want to convert.
- Click the "Upload" button.



- Once the file is uploaded, the Lambda function integrated with the bucket will automatically trigger and execute the conversion code.
- The output of the conversion process will be available in the destination bucket that you specified in the Lambda function code.



- As well, once you check the job completed or getting error in mediaconvert console.
- Open the MediaConvert jobs page and find a job for the input 'test.mp4' that was started near the time your upload completed.



- Note the time that the upload is completed.

Ensure that you've configured the Lambda function and S3 bucket permissions correctly for this test to be successful.

UPDATE THE EXISTING LAMBDA IAM ROLE FOR MEDIAPACKAGE:

- Open the AWS Management Console and go to the IAM service.
- In the IAM dashboard, click on "Policies" in the left sidebar then, click the create policy in the right corner.
- Choose the "JSON" tab.

- Copy and paste the following policy JSON in the editor:



```
1  {
2    "Version": "2012-10-17",
3    "Statement": [
4      {
5        "Effect": "Allow",
6        "Action": "mediapackage-vod:*",
7        "Resource": "*"
8      }
9    ]
10 }
```

- Review the policy to ensure it grants the necessary permissions for MediaPackage VOD.
- Give your policy a name and description that clearly identifies its purpose, e.g., "MediaPackageVODAccessPolicy".
- Click on "Review policy" to proceed to the next step.
- Review the policy details and click "Create policy".
- Once the policy is created, go to the role and choose the lambda role which was created early.
- Click on "Attach permissions policies".
- Search for the policy you just created (e.g., "MediaPackageVODAccessPolicy").
- For this scenario, you'll need to attach the following pre-built policies:
 1. **AmazonDynamoDBFullAccess:** Allows reading from and writing to any DynamoDB table.
 2. **AWSElementalMediaConvertFullAccess:** Provides full access to AWS Elemental MediaConvert for video conversion.
 3. **CloudFrontFullAccess:** Provides full access to CloudFront for content delivery management.
 4. **SecretsManagerReadWrite:** Allows reading and writing of secrets in AWS Secrets Manager.
- Select the policy and click "Attach policy". Verify that the policy is attached to your Lambda role by checking the role's details.
- After completing these steps, your Lambda function should have the necessary permissions to access MediaPackage VOD.

ALLOWING AWS ELEMENTAL MEDIAPACKAGE TO ACCESS OTHER AWS SERVICES

Some features require you to allow MediaPackage to access other AWS services, such as Amazon S3 and AWS Secrets Manager (Secrets Manager). To allow this access, create an IAM role and policy with the appropriate permissions. The following steps describe how to create roles and policies for MediaPackage features.

Topics

- Step 1: Create a policy
- Step 2: Create a role
- Step 3: Modify the trust relationship

Step 1: CREATE A POLICY

The IAM policy defines the permissions that AWS Elemental MediaPackage (MediaPackage) requires to access other services.

- For video on demand (VOD) workflows, create a policy that allows MediaPackage to read from the Amazon S3 bucket, verify the billing method, and retrieve content. For the billing method, MediaPackage must verify that the bucket does not require the requester to pay for requests. If the bucket has requestPayment enabled, MediaPackage can't ingest content from that bucket.
- For live-to-VOD workflows, create a policy that allows MediaPackage to read from the Amazon S3 bucket and store the live-to-VOD asset in it.
- For content delivery network (CDN) authorization, create a policy that allows MediaPackage to read from a secret in Secrets Manager.

The following sections describe how to create these policies.

Topics:

- CloudFront access for MediaPackage
- Amazon S3 access for VOD workflows
- Secrets Manager access for CDN authorization

CLOUDFRONT ACCESS FOR MEDIAPACKAGE:

To use the JSON policy editor to create a policy

1. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation pane on the left, choose Policies.
3. At the top of the page, choose Create policy.
4. In the Policy editor section, choose the JSON option.
5. Enter the following JSON policy document:



```
1  {
2    "Version": "2012-10-17",
3    "Statement": [
4      {
5        "Effect": "Allow",
6        "Action": [
7          "cloudfront:GetDistribution",
8          "cloudfront:UpdateDistribution",
9          "cloudfront:CreateDistribution",
10         "cloudfront:TagResource",
11         "tag:GetResources"
12       ],
13       "Resource": "*"
14     }
15   ]
16 }
```

6. Choose Next.
7. On the Review and create page, enter a Policy name and a Description (optional) for the
8. policy that you are creating. Review Permissions defined in this policy to see the permissions that are granted by your policy.
9. Choose Create policy to save your new policy.

AMAZON S3 ACCESS FOR VOD WORKFLOWS:

If you're using MediaPackage to ingest a VOD asset from an Amazon S3 bucket and to package and deliver that asset, you need a policy that allows you to do these things in Amazon S3:

- **GetObject** - MediaPackage can retrieve the VOD asset from the bucket.
- **GetBucketLocation** - MediaPackage can retrieve the Region for the bucket. The bucket must be in the same region as the MediaPackage VOD resources.

- `GetBucketRequestPayment` - `MediaPackage` can retrieve the payment request information. `MediaPackage` uses this information to verify that the bucket doesn't require the requester to pay for the content requests.

To use the JSON policy editor to create a policy

1. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation pane on the left, choose Policies. Choose Get Started.
3. At the top of the page, choose Create policy.
4. In the Policy editor section, choose the JSON option.
5. Choose Next.



```
1 {
2   "Version": "2012-10-17",
3   "Statement": [
4     {
5       "Action": [
6         "s3:GetObject",
7         "s3:GetBucketLocation",
8         "s3:GetBucketRequestPayment",
9         "s3:ListBucket"
10      ],
11      "Resource": [
12        "arn:aws:s3:::bucket_name/*",
13        "arn:aws:s3:::bucket_name"
14      ],
15      "Effect": "Allow"
16    }
17  ]
18 }
```

6. On the Review and create page, enter a Policy name and a Description (optional) for the policy that you are creating. Review Permissions defined in this policy to see the permissions that are granted by your policy.
7. Choose Create policy to save your new policy.

SECRETS MANAGER ACCESS FOR CDN AUTHORIZATION:

If you use content delivery network (CDN) authorization headers to restrict access to your endpoints in `MediaPackage`, you need a policy that allows you to do these things in `Secrets Manager`:

- `GetSecretValue` - `MediaPackage` can retrieve the encrypted authorization code from a version of the secret.
- `DescribeSecret` - `MediaPackage` can retrieve the details of the secret, excluding encrypted fields.
- `ListSecrets` - `MediaPackage` can retrieve a list of secrets in the AWS account.

- ListSecretVersionIds: MediaPackage can retrieve all of the versions that are attached to the specified secret.

To use the JSON policy editor to create a policy

1. Sign in to the AWS Management Console and open the IAM console at <https://console.aws.amazon.com/iam/>.
2. In the navigation column on the left, choose Policies.
3. At the top of the page, choose Create policy.
4. Choose the JSON tab.
5. Enter the following JSON policy document, replacing region, account-id, secret-name, and role-name with your own information:

```
1  {
2    "Version": "2012-10-17",
3    "Statement": [
4      {
5        "Effect": "Allow",
6        "Action": [
7          "secretsmanager:GetSecretValue",
8          "secretsmanager:DescribeSecret",
9          "secretsmanager:ListSecrets",
10         "secretsmanager:ListSecretVersionIds"
11        ],
12        "Resource": [
13          "arn:aws:secretsmanager:region:account-id:secret:secret-name"
14        ]
15      },
16      {
17        "Effect": "Allow",
18        "Action": [
19          "iam:GetRole",
20          "iam:PassRole"
21        ],
22        "Resource": "arn:aws:iam::account-id:role/role-name"
23      }
24    ]
25  }
```

6. Choose Review policy.
7. On the Review policy page, enter a Name and an optional Description for the policy that
8. you are creating. Review the policy Summary to see the permissions that are granted by your
9. policy. Then choose Create policy to save your work.

Step 2: CREATE A ROLE

CREATE AN IAM ROLE FOR MEDIA PACKAGE:

- Open the AWS Management Console and go to the IAM service.
- In the IAM dashboard, click on "Roles" in the left sidebar, then click on the "Create role" button.

- Choose "AWS service" as the trusted entity, and select "Lambda" as the service that will use this role.
- Click "Next: Permissions" to proceed to the permissions configuration.
- In the "Attach permissions policies" step, search for and select the policies that grant the necessary permissions. For this scenario, you'll need to attach the following pre-build policies:

AmazonS3FullAccess: Grants full access to Amazon S3 buckets.

- As well, you add the policies you just created in Step 1: Create a policy eg:
 1. CloudFront access for MediaPackage
 2. Amazon S3 access for VOD workflows
 3. Secrets Manager access for CDN authorization
- Review the policies attached to the role to ensure they grant the necessary permissions.
- Give the role a descriptive name that reflects its purpose, such as "VODLambdaRole" for the Role name.
- Choose Create role.

CREATE AN IAM ROLE FOR SECRET MANAGER:

- Open the AWS Management Console and go to the IAM service.
- In the IAM dashboard, click on "Roles" in the left sidebar, then click on the "Create role" button.
- Choose "AWS service" as the trusted entity, and select "Lambda" as the service that will use this role.
- Click "Next: Permissions" to proceed to the permissions configuration.
- In the "Attach permissions policies" step, search for and select the policies that grant the necessary permissions. For this scenario, you'll need to attach the secret manager policy that you just created in Step 1: Create a policy.
 - Secrets Manager access for CDN authorization
- Review the policies attached to the role to ensure they grant the necessary permissions.
- Give the role a descriptive name that reflects its purpose, such as "VODLambdaRole" for the Role name.
- Choose Create role.

Step 3: MODIFY THE TRUST RELATIONSHIP

The trust relationship defines what entities can assume the role that you created in the section called “Step 2: Create a role”. When you created the role and established the trusted relationship, you chose Amazon EC2 as the trusted entity. Modify the role so that the trusted relationship is between your AWS account and AWS Elemental MediaPackage.

To change the trust relationship to MediaPackage

1. Access the role that you created in Step 2: Create a role.
2. On the Summary page for the role, choose Trust relationships.
3. Choose Edit trust relationship.
4. On the Edit Trust Relationship page, in the Policy Document, change **ec2.amazonaws.com**
5. to **mediapackage.amazonaws.com**.

The policy document should now look like this:



```
1  {
2    "Version": "2012-10-17",
3    "Statement": [
4      {
5        "Effect": "Allow",
6        "Principal": {
7          "Service": "mediapackage.amazonaws.com"
8        },
9        "Action": "sts:AssumeRole"
10     }
11   ]
12 }
```

6. Choose Update Trust Policy.
7. On the Summary page, make a note of the value in Role ARN. You use this ARN when you ingest source content for video on demand (VOD) workflows. The ARN looks like this:

arn:aws:iam::111122223333:role/role-name

In the example, 111122223333 is your AWS account number.

CDN authorization in AWS Elemental MediaPackage:

Content Delivery Network (CDN) authorization helps you to protect your content from unauthorized use. When you configure CDN authorization, MediaPackage only fulfills playback requests that are authorized

between MediaPackage and your CDN. This prevents users from bypassing the CDN in order to directly access your content on the origin.

How it works

You configure your CDN, such as Amazon CloudFront, to include a custom HTTP header in content requests to MediaPackage.

Custom HTTP header and example value.

```
X-MediaPackage-CDNIdentifier: 9ceebbe7-9607-4552-8764-876e47032660
```

You store the header value as a secret in AWS Secrets Manager. When your CDN sends a playback request, MediaPackage verifies that the secret's value matches the custom HTTP header value.

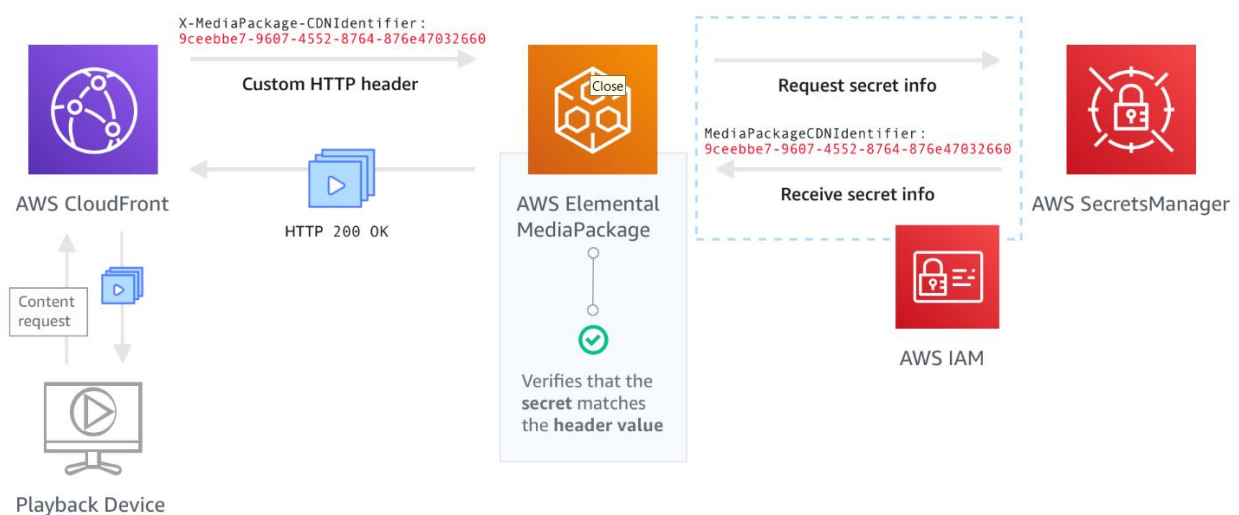
MediaPackage is given permission to read the secret with an AWS Identity and Access Management permissions policy and role.

Secret key and example value.

```
{"MediaPackageCDNIdentifier": "9ceebbe7-9607-4552-8764-876e47032660"}
```

If the values match, MediaPackage serves the content along with an HTTP 200 OK status code. If it's not a match, or if the authorization request fails, then MediaPackage doesn't serve the content, and sends an HTTP 403 Unauthorized status code.

The following image shows successful CDN authorization using Amazon CloudFront.



SETTING UP CDN AUTHORIZATION:

Complete the following steps to set up CDN authorization.

Topics

- Step 1: Configure a CDN custom origin HTTP header
- Step 2: Store the value as a secret in AWS Secrets Manager

Step 1: Configure a CDN custom origin HTTP header

In your CDN, configure a custom origin HTTP header that contains the header **X-MediaPackage-CDNIdentifier** and a value. For the value, we recommend that you use the UUID version 4 format, which produces a 36-character string. If you aren't using the UUID version 4 format, the value must be 8-128 characters long.

Example header and value

```
X-MediaPackage-CDNIdentifier: 9ceebbe7-9607-4552-8764-876e47032660
```

To create a custom header in Amazon CloudFront

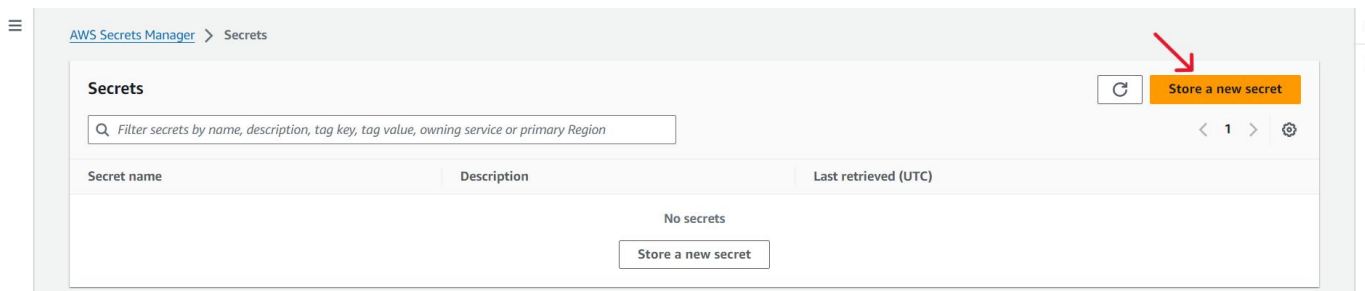
1. Sign in to the AWS Management Console and open the CloudFront console at <https://console.aws.amazon.com/cloudfront/v4/home>.
2. Create or edit a distribution.
3. In Origin Settings, complete the fields. You will use this same value for your secret in SecretsManager.
 - For Header Name, enter X-MediaPackage-CDNIdentifier.
 - For Value, enter a value. We recommend that you use UUID version 4 format, which produces a 36-character string. If you aren't using the UUID version 4 format, the value must be 8-128 characters long.
4. Complete the rest of the fields and save the distribution.

Step 2: Store the value as a secret in AWS Secrets Manager

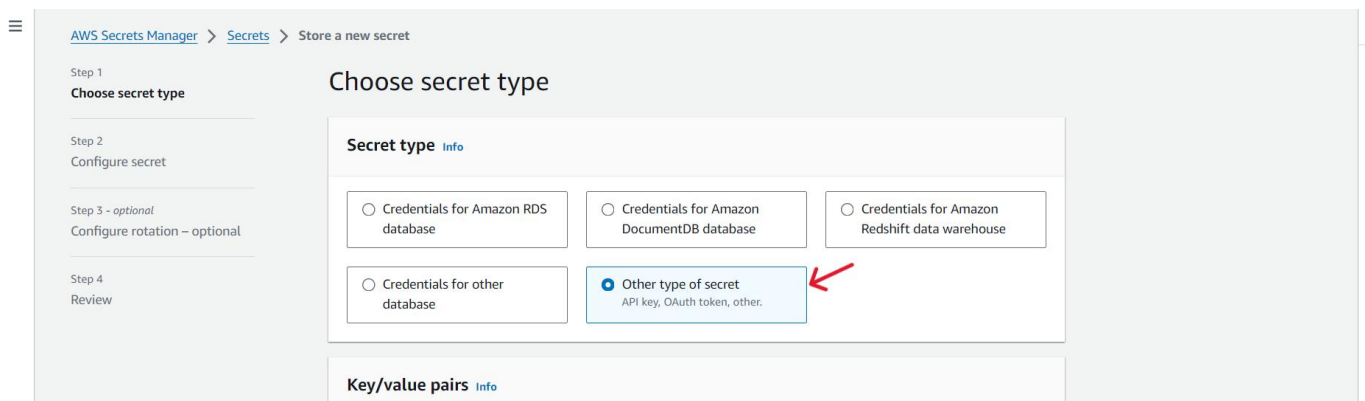
Store the same value that you use in your custom origin HTTP header as a secret in AWS Secrets Manager. The secret must use the same AWS account and Region settings as your AWS Elemental MediaPackage resources. MediaPackage doesn't support sharing secrets across accounts or Regions. However, you can use the same secret across multiple endpoints in the same Region and on the same account.

To store a secret in Secrets Manager

1. Sign in to the AWS Secrets Manager console at <https://console.aws.amazon.com/secretsmanager/>.
2. Choose Store a new secret.



3. For Secret type, choose Other type of secrets.



4. For Key/value pairs, enter the key and value information.

- In the box on the left, enter MediaPackageCDNIdentifier.
- In the box on the right, enter the value that you configured for your custom origin HTTP header.
For example, 9ceebbe7-9607-4552-8764-876e47032660.

5. For Encryption key, you can keep the default value as DefaultEncryptionKey.
6. Choose Next.

Key/value pairs [Info](#)

Key/value | Plaintext

MediaPackageCDNIdentifier	0a1238e9-2bc6-4573-aa0e-c5ac96638e8f
---------------------------	--------------------------------------

[+ Add row](#)

Encryption key [Info](#)

You can encrypt using the KMS key that Secrets Manager creates or a customer-managed KMS key that you create.

aws/secretsmanager [Add new key](#)

Cancel **Next**

- For Secret name, we recommend that you prefix it with MediaPackage/ so that you know it's a secret used for MediaPackage. For example, MediaPackage/cdn_auth_us-west-2.

Step 1 [Choose secret type](#)

Step 2 **Configure secret**

Step 3 - optional
Configure rotation - optional

Step 4

Configure secret

Secret name and description [Info](#)

Secret name
A descriptive name that helps you find your secret later.

MediaPackage/cdn_auth_ap-south-1

Secret name must only contain alphanumeric characters and the characters /_+=.@-

Cancel **Next**

- Choose Next.

Replicate secret - optional

Create read-only replicas of your secret in other regions. Replica secrets incur a charge.

[Learn more in the User Guide.](#)

Choose a region to create a read-only replica of your secret. [Info](#)

Replica secrets contain the same secret value and metadata.

AWS region **Encryption key**

aws/secretsmanager

[Add Region](#)

Cancel Previous **Next**

- For Configure automatic rotation, keep the default Disable automatic rotation setting.
- Choose Next

Rotation function [Info](#)

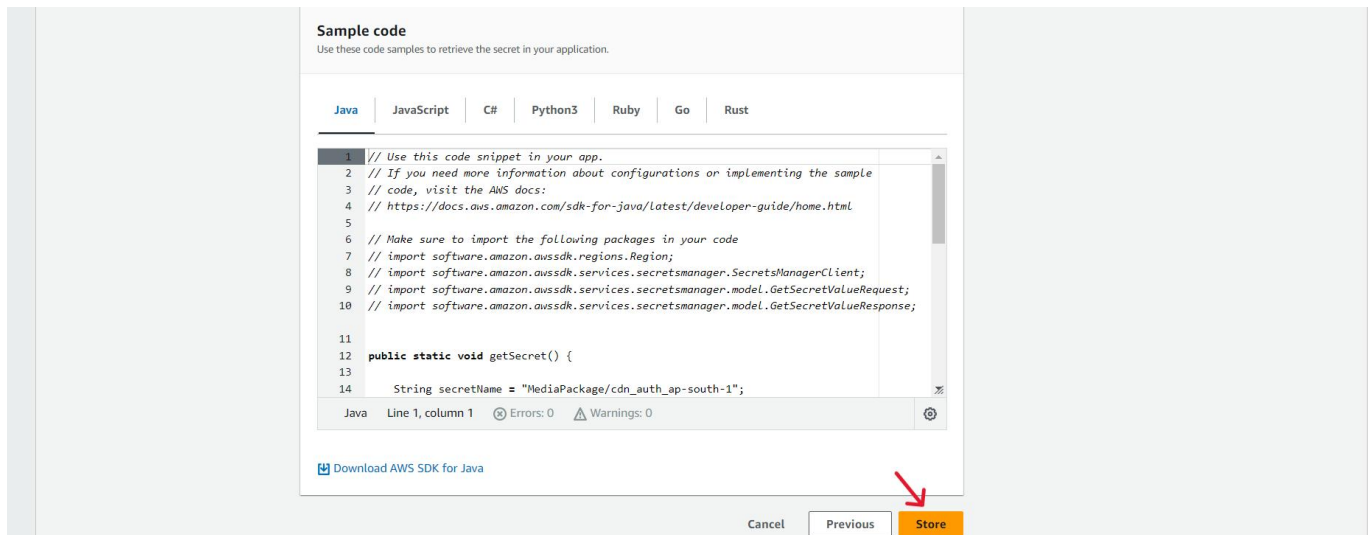
Lambda rotation function [Info](#)

Choose a Lambda function that can rotate this secret.

media-convert-function [Create function](#)

Cancel Previous **Next**

- Then, Click the store button.

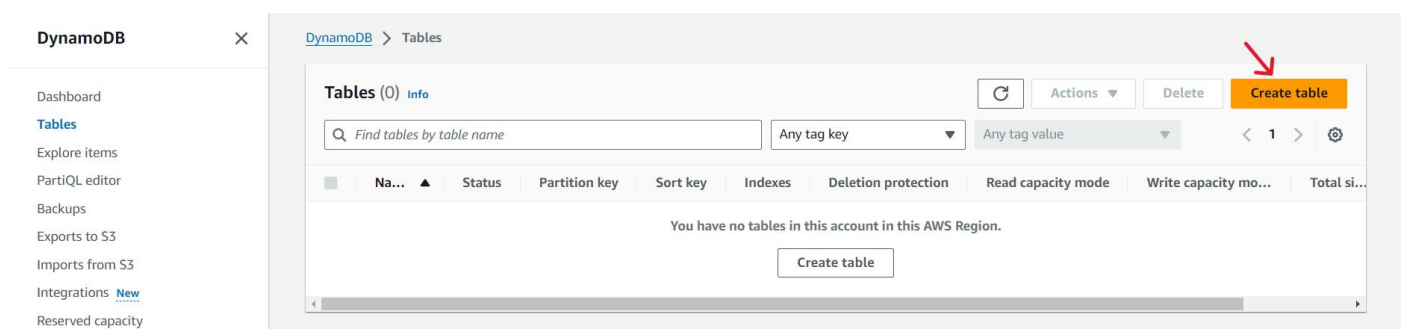


12. Select your secret name to view the Secret ARN. The ARN has a value similar to `arn:aws:secretsmanager:us-west-2:123456789012:secret:MediaPackage/cdn_auth_test-xxxxxx`. You use the Secret ARN when you configure CDN authorization for MediaPackage.

CREATE A TABLE IN DYNAMODB FOR STORING METADATA:

To create a database (called a "table" in DynamoDB) with the specified attributes in Amazon DynamoDB, follow these steps:

- Navigate to the AWS Management Console at <https://aws.amazon.com/console/> and sign in to your AWS account.
- Once signed in, search for "DynamoDB" in the AWS Management Console search bar or navigate to the "Databases" section and select "DynamoDB".
- In the DynamoDB dashboard, click on the "Create table" button to initiate the table creation process.



- Enter a name for your table, such as "VideoMetadata" or any other relevant name.
- Specify the primary key for your table. In this case, you'll want to use "sno" as the partition key.
- Optionally, you can define a sort key if needed for additional sorting capabilities.

DynamoDB > Tables > Create table

Create table

Table details

Info

DynamoDB is a schemaless database that requires only a table name and a primary key when you create the table.

Table name

This will be used to identify your table.

media-services

Between 3 and 255 characters, containing only letters, numbers, underscores (_), hyphens (-), and periods (.).

Partition key

The partition key is part of the table's primary key. It is a hash value that is used to retrieve items from your table and allocate data across hosts for scalability and availability.

s.no

String

1 to 255 characters and case sensitive.

Sort key - optional

You can use a sort key as the second part of a table's primary key. The sort key allows you to sort or search among all items sharing the same partition key.

Enter the sort key name

String

1 to 255 characters and case sensitive.

- Configure any additional settings for your table, such as provisioned capacity (for throughput) or auto scaling options. For simplicity, you can start with the default settings.

Table settings

Default settings

The fastest way to create your table. You can modify these settings now or after your table has been created.

Customize settings

Use these advanced features to make DynamoDB work better for your needs.

Default table settings

These are the default settings for your new table. You can change some of these settings after creating the table.

Setting	Value	Editable after creation
Table class	DynamoDB Standard	Yes
Capacity mode	Provisioned	Yes
Provisioned read capacity	5 RCU	Yes
Provisioned write capacity	5 WCU	Yes

- Define the attributes for your table. In this case, you'll have four attributes: "sno" (as the partition key), "filename", "domainname", and "endpoints".
- Specify the data types for each attribute. For example, "sno" might be a Number, while "filename", "domainname", and "endpoints" could be Strings.
- Review the table configuration settings to ensure they align with your requirements.
- Click on the "Create table" button to finalize the creation of your DynamoDB table.

Tags

Tags are pairs of keys and optional values, that you can assign to AWS resources. You can use tags to control access to your resources or track your AWS spending.

No tags are associated with the resource.

Add new tag

You can add 50 more tags.

Cancel

Create table

- Once the table is created, you can access it from the DynamoDB dashboard to insert, query, update, and delete items.
- Use the AWS SDK or AWS CLI to interact with the table programmatically from your applications or scripts.

CREATE AN ANOTHER FUNCTION IN AWS LAMBDA FOR MEDIAPACKAGE:

- Sign in to your AWS account at <https://aws.amazon.com/console/>.
- Navigate to the Lambda service by searching for "Lambda" in the AWS Management Console.
- In the Lambda console, click on the "Create function" button.
- Choose the "Author from scratch" option to create a new function from scratch.
- Configure Basic Information:
 - Enter a name for your function in the "Function name" field.
 - Select the runtime environment for your function (e.g., Python, Node.js, Java).
 - Choose an existing execution role or create a new one with necessary permissions.
- Once you've configured the basic information, click on the "Create function" button to create your Lambda function.
- After creating lambda function you need to configure the S3 Bucket (MediaConvert Destination Bucket) with the current function.

Integrate the S3 destination bucket with a lambda function:

- In the lambda function console, navigate to the left middle of top and click on "Add trigger." Choose S3 as the trigger type.
 - Specify the S3 bucket (**MediaConvert Destination Bucket**) and the event type (e.g., ObjectCreated) that should trigger the Lambda function.
 - Ensure that the Lambda function has the necessary permissions to access the S3 bucket. This typically involves creating an IAM role with permissions to read from the S3 bucket and attaching it to the Lambda function.
 - When the media convert lambda function stored the converted files as an object to the destination S3 bucket and verify that the media package lambda function is triggered automatically.
- In the function code editor, you see the sample return type program.
 - After lambda integration with S3. You can start to write the code to execute media convert.
 - For your reference, you can replace the following code instead of sample return type program in lambda function.

lambda_funation.py

```
import boto3
import json
import uuid
import re
from botocore.exceptions import ClientError
```

```

# Function to create a packaging group
def create_packaging_group(packaging_group_name, CDNIdentifierSecretARN, SecretRoleARN, secret_name, distribution_id):
    media_package = boto3.client('mediapackage-vod')

    try:
        # Check if the packaging group already exists
        response = media_package.describe_packaging_group(Id=packaging_group_name)
        return response['DomainName'] # Return the domain name if it exists

        # Print a message if the packaging group already exists
        print("Packaging Group Already Exist")

    except media_package.exceptions.NotFoundException:
        try:
            # Create the packaging group if it doesn't exist
            response = media_package.create_packaging_group(
                Authorization={
                    'CdnIdentifierSecret': CDNIdentifierSecretARN,
                    'SecretsRoleArn': SecretRoleARN,
                },
                Id=packaging_group_name
            )
            print("Packaging Group created successfully")

            # Create packaging configuration for the new group
            create_packaging_configuration(packaging_group_name)
            print("Packaging configuration created successfully")

            # Retrieve secret value
            secret_value = retrieve_secrets_value(secret_name)

            # Create distribution in CloudFront
            distribution_domain_name = create_distribution(distribution_id, packaging_group_name,
                response['DomainName'], secret_value)
            print(distribution_domain_name)
            print("Origin created successfully in CloudFront")

            return response['DomainName']

        except Exception as e:
            print("Error creating Packaging Group:", e)

# Function to retrieve secret value from Secrets Manager
def retrieve_secrets_value(secret_name):
    secrets_manager_client = boto3.client('secretsmanager')
    response = secrets_manager_client.get_secret_value(SecretId=secret_name)

    if 'SecretString' in response:
        secret_string = response['SecretString']
        secret_dict = json.loads(secret_string)
        for key, value in secret_dict.items():
            print(f"Key: {key}, Value: {value}")
        return value

```

```

# Function to create distribution in CloudFront
def create_distribution(distribution_id, packaging_group_name, domain_name, secret_value):
    cloudfront_client = boto3.client('cloudfront')
    id = packaging_group_name+'-distribution'
    parts = domain_name.split("/")
    domain_name = parts[1]
    headers = { 'X-MediaPackage-CDNIdentifier' : secret_value}

    # Fetch the existing distribution configuration
    response = cloudfront_client.get_distribution_config(Id=distribution_id)
    existing_config = response['DistributionConfig']
    existing_etag = response['ETag']

    # Update the distribution configuration to add the new origin
    existing_config['Origins']['Quantity'] += 1 # Increment the quantity of origins
    existing_config['Origins']['Items'].append({
        'Id': packaging_group_name,
        'DomainName': domain_name,
        'OriginPath': '',
        'CustomOriginConfig': {
            'HTTPPort': 80,
            'HTTPSPort': 443,
            'OriginProtocolPolicy': 'match-viewer',
            'OriginReadTimeout': 30,
            'OriginKeepaliveTimeout': 5,
            'OriginSslProtocols': {
                'Quantity': 1,
                'Items': ['TLSv1']
            },
        },
    },
    ),
    'CustomHeaders': {
        'Quantity': 1,
        'Items': [
            {
                'HeaderName': 'X-MediaPackage-CDNIdentifier',
                'HeaderValue': secret_value
            }
        ]
    }
})

# Update the distribution with the modified configuration
response = cloudfront_client.update_distribution(
    DistributionConfig=existing_config,
    Id=distribution_id,
    IfMatch=existing_etag
)

# Check the response for success
if response['ResponseMetadata']['HTTPStatusCode'] == 200:
    print("Distribution configuration updated successfully")
else:

```



```

        print("Failed to update distribution configuration")
        return None

# Function to retrieve domain name of distribution from CloudFront
def retrieve_domain_name(distribution_id):
    cloudfront_client = boto3.client('cloudfront')

    try:
        response = cloudfront_client.get_distribution(Id=distribution_id)
        distribution_config = response['Distribution']
        domain_name = distribution_config['DomainName']
        return domain_name
    except Exception as e:
        print("Error retrieving distribution domain name:", e)
        return None

# Function to create packaging configuration
def create_packaging_configuration(packaging_group_name):
    media_package = boto3.client('mediapackage-vod')

    response = media_package.create_packaging_configuration(
        HlsPackage={
            'HlsManifests': [
                {
                    'AdMarkers': 'NONE',
                    'ManifestName': 'index',
                    'ProgramDateTimeIntervalSeconds': 0,
                },
            ],
            'SegmentDurationSeconds': 10,
        },
        Id= "Yt-HLS",
        PackagingGroupId= packaging_group_name,
    )

    return response['Arn']

# Function to ingest video to packaging group
def ingest_video_to_packaging_group(packaging_group_name, asset_id, s3_bucket, s3_key, mp_arn):
    media_package = boto3.client('mediapackage-vod')

    try:
        response = media_package.describe_asset(Id=asset_id)
        return response['EgressEndpoints'][0]['Url']
        print("Asset Already Exist")

    except media_package.exceptions.NotFoundException:
        try:
            response = media_package.create_asset(
                Id=asset_id,
                PackagingGroupId=packaging_group_name,
                SourceArn=f'arn:aws:s3:::{s3_bucket}/{s3_key}',
                SourceRoleArn=mp_arn

```

```

    )

    return response['EgressEndpoints'][0]['Url']

except Exception as e:
    print(e)

# Function to get ingested video endpoint
def get_ingested_video_endpoint(asset_arn):
    media_package = boto3.client('mediapackage')

    response = media_package.list_assets(
        MaxResults=1,
        PackagingGroupId='',
        NextToken='',
        SortBy='CREATION_TIME'
    )

    asset = response['Assets'][0]
    endpoints = media_package.list_asset_endpoints(
        Id=asset['Id']
    )

    return endpoints['Endpoints'][0]['Url']

# Function to get key from S3 event
def get_key(event, event_key):
    pattern = r'^([^_]+)(360|540|720)\.m3u8'
    parts = event_key.split('.')
    file_extension = parts[-1]
    s3_key = ''
    if re.search(pattern, event_key):
        pass
    else:
        if(file_extension == "m3u8"):
            s3_key = event['Records'][0]['s3']['object']['key']
        else:
            raise Exception("Key doesn't match any valid pattern")

    return s3_key

# Function to check if URL exists in DynamoDB
def check_url_exists(tablename, endpoint):
    dynamodb = boto3.client('dynamodb')

    try:
        # Construct the Key parameter as a dictionary
        key = {'endpoint': {'S': endpoint}}
        # Call get_item with the corrected Key parameter
        response = dynamodb.get_item(TableName=tablename, Key=key)
        if 'Item' in response:
            return response['Item']['url']['S']
    except ClientError as e:

```

```

    print(e.response['Error']['Message'])
    return False

# Function to manage DynamoDB entries
def mange_dynamodb(tablename, filename, endpoint, domain_name):
    dynamodb = boto3.client('dynamodb')

    if endpoint is not None: # Check if endpoint is not None before proceeding
        if not check_url_exists(tablename, endpoint):
            # Use a scan operation with Count parameter to count the items
            response = dynamodb.scan(TableName=tablename, Select='COUNT')
            # Retrieve the count of items
            item_count = response['Count']
            response = dynamodb.put_item(
                TableName=tablename,
                Item={
                    's.no': {'S': str(item_count + 1)},
                    'filename': {'S': filename},
                    'endpoint': {'S': endpoint},
                    'domainname': {'S': domain_name}
                }
            )
            print("Endpoint added to the database.")
            return "Successfully Completed"
        else:
            print("Endpoint already exists in the database.")
    else:
        print("Endpoint is None. Cannot add to the database.")

# Main Lambda handler
def lambda_handler(event, context):
    s3_bucket = event['Records'][0]['s3']['bucket']['name']
    event_key = event['Records'][0]['s3']['object']['key']
    s3_key = get_key(event, event_key)

    parts = s3_key.split('/')
    filename = parts[-1].split('.')
    asset_id = filename[0]
    name = filename[0]

    packaging_group_name = parts[0]

    mp_arn = "Your-Media-Package-ARN"

    unique_id = str(uuid.uuid4())

    tablename = "Your-Table-Name"

    secret_name = "Your-Secret-Name"

    distribution_id = "Your-Distribution-Id"

    CDNIdentifierSecretARN = "Your-Secret-ARN"

```

```

SecretRoleARN = "Your-Secret-Role-ARN"

packaging_group_domain_name = create_packaging_group(packaging_group_name, CDNIdentifierSecretARN, SecretRoleARN,
secret_name, distribution_id)

endpoint_url = ingest_video_to_packaging_group(packaging_group_name, asset_id, s3_bucket, s3_key, mp_arn)

cdn_domain_name = retrieve_domain_name(distribution_id)

result = mange_dynamodb(tablename, name, endpoint_url, cdn_domain_name)
print(result)

return {
    'statusCode': 200,
    'body': {
        'status': "success"
    }
}

```

CODE WALKTHROUGH:

Importing Libraries:

The code starts by importing necessary libraries like boto3 for AWS SDK, json for JSON handling, uuid for generating unique identifiers, re for regular expressions, and ClientError from botocore.exceptions for handling AWS client errors.

Function Definitions:

- **create_packaging_group:** This function creates a packaging group in AWS Elemental MediaPackage.
- **retrieve_secrets_value:** Retrieves the value of a secret stored in AWS Secrets Manager.
- **create_distribution:** Creates a distribution in Amazon CloudFront.
- **retrieve_domain_name:** Retrieves the domain name of a CloudFront distribution.
- **create_packaging_configuration:** Creates a packaging configuration in AWS Elemental MediaPackage.
- **ingest_video_to_packaging_group:** Ingests a video into a packaging group in AWS Elemental MediaPackage.
- **get_ingested_video_endpoint:** Retrieves the endpoint URL of an ingested video.
- **get_key:** Extracts the key from an S3 event based on a specified pattern.
- **check_url_exists:** Checks if a URL exists in a DynamoDB table.
- **manage_dynamodb:** Manages data in a DynamoDB table.
- **lambda_handler:** The main Lambda function handler.

Lambda Handler Function:

- Retrieves necessary information from the S3 event such as bucket name and object key.
- Constructs various parameters required for the AWS services like MediaPackage, CloudFront, and DynamoDB.
- Calls create_packaging_group to create a packaging group and associated resources.

- Ingests the video into the packaging group using `ingest_video_to_packaging_group`.
- Retrieves the CloudFront distribution domain name.
- Manages DynamoDB table to store information about the ingested video.
- Returns a response indicating the success status.

Error Handling:

- Various try-except blocks are used to catch exceptions that might occur during AWS API calls.
- Error messages are printed for debugging purposes.

Comments: Comments are added throughout the code to explain the purpose of each function, parameter, and logical step.

Function Calls: Function calls are made in a logical sequence to perform the required operations.

Response: Finally, a response object is returned with a success status and message.

TEST THE LAMBDA FUNCTION:

- Go to the S3 bucket you created earlier for input.
- Click on the "Add file" button.
- Select the video file you want to convert.
- Click the "Upload" button.
- Once the file is uploaded, the Lambda function integrated with the bucket will automatically trigger and execute the conversion code.
- The output of the conversion process will be available in the destination bucket that you specified in the Lambda function (MediaConvert) code.
- By continuing this, when the conversion object is stored in the destination bucket, the integrated another lambda function (MediaPackage) will be triggered automatically.
- The Lambda function can trigger further actions:
 - Create a packaging groups and configure packaging settings.
 - Create assets.
 - Set up CloudFront distribution for content delivery.
 - Retrieve the secret value from the secret manager.
 - Store metadata like filename, domain name (CloudFront), and endpoints in DynamoDB.

REQUIREMENTS INSTALLING

INSTALLING XAMPP SERVER:

Download XAMPP:

- Go to the XAMPP official website and download the XAMPP installer for your operating system (Windows, macOS, Linux).

Run the Installer:

- Once the download is complete, run the installer and follow the installation wizard.
- Choose the components you want to install (e.g., Apache, MySQL, PHP, phpMyAdmin).
- Select the installation directory (the default directory is usually fine).

Start Apache and MySQL:

- After the installation is complete, start the Apache and MySQL services using the XAMPP Control Panel.
- On Windows, you can find the XAMPP Control Panel in the Start menu or launch it directly from the installation directory.

Verify Installation:

- Open your web browser and navigate to <http://localhost>. You should see the XAMPP dashboard, indicating that the server is running correctly.

INSTALLING COMPOSER:

Download Composer:

- Go to the Composer official website and download the Composer installer appropriate for your operating system (Windows, macOS, Linux).

Run the Installer:

- Run the downloaded installer and follow the installation instructions.
- Composer usually requires PHP to be installed on your system, which is included in XAMPP.

Verify Installation:

- Open a command-line interface (Terminal on macOS/Linux, Command Prompt on Windows).
- Type **composer --version** and press Enter. You should see the Composer version number printed, confirming that Composer is installed correctly.

ADDITIONAL NOTES:

Environment Variables (Windows):

- If you're using Windows, make sure to add the paths to XAMPP and Composer to your system's PATH environment variable. This allows you to run XAMPP and Composer commands from any directory.

Updating Composer:

- Composer is frequently updated with new features and bug fixes. You can update Composer to the latest version by running **composer self-update** in the command-line interface.

Starting and Stopping XAMPP:

- You can start and stop the XAMPP services (Apache, MySQL) using the XAMPP Control Panel. Make sure to stop the services when you're not using them to free up system resources.

By following these steps, you'll have XAMPP server and Composer installed on your local machine, ready for web development projects.

CREATE A USER INTERFACE FOR DELIVER THE PROCESSED VIDEO CONTENT:

- Create the project environment in the xampp server That means the project folder should be in **C://xampp/htdocs/projectfolder/index.php**. Because the php is a server side programming language thus, it needs an server to execute the code.
- The following code represents, user interface for delivering the processed video content to the user.
- Here, we use HTML, CSS, and JavaScript to design the user interface and video.js are used to embed the video to the web page.
- Ensure you have the AWS SDK for PHP installed and configured in your project. You can install it using Composer or by downloading the SDK directly.
- Use the AWS SDK for PHP to query DynamoDB to retrieve metadata about processed videos.
- Metadata may include information like filename, CloudFront domain, and endpoint.

index.php

```
<?php

require 'vendor/autoload.php';

use Aws\DynamoDb\DynamoDbClient;
use Aws\Exception\AwsException;

// Specify your AWS credentials and region
$credentials = new Aws\Credentials\Credentials('YOUR_ACCESS_KEY_ID', 'YOUR_SECRET_ACCESS_KEY');
$region = 'YOUR_AWS_REGION';

// Create a DynamoDB client
$dynamodb = new DynamoDbClient([
    'region' => $region,
    'version' => 'latest',
```

```

        'credentials' => $credentials
    ));

$tableName = 'YOUR_TABLE_NAME';

$params = [
    'TableName' => $tableName,
    'ProjectionExpression' => 'domainname, endpoint, filename'
];

try {
    $result = $dynamodb->scan($params);

    $domain = [];
    $filenames = [];
    foreach ($result['Items'] as $item) {
        $endpoint = $item['endpoint']['S'];
        $domainName = $item['domainname']['S'];
        $filename = $item['filename']['S'];

        $parts = explode('amazonaws.com', $endpoint);

        $url = "https://" . $domainName . $parts[1];

        $domain[] = $url;

        $names = $filename;

        $filenames[] = $names;
    }
} catch (AwsException $e) {
    echo $e->getMessage();
}
?>

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>YouTube.com</title>

    <link rel="preconnect" href="https://fonts.googleapis.com">
    <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
    <link href="https://fonts.googleapis.com/css2?family=Roboto:wght@500;700&display=swap" rel="stylesheet">

    <!-- Videojs CDN -->
    <link href="https://vjs.zencdn.net/8.10.0/video-js.css" rel="stylesheet"/>
    <script defer src="https://vjs.zencdn.net/8.10.0/video.min.js"></script>

    <link href="https://unpkg.com/@silvermine/videojs-quality-selector/dist/css/quality-selector.css" rel="stylesheet">
    <script src="https://unpkg.com/@silvermine/videojs-quality-selector/dist/js/silvermine-videojs-quality-selector.min.js"></script>

</head>

<body>

<main>
    <section class="video-grid">
        <?php

            // Loop through the endpoints array and generate HTML dynamically

```



```

        for ($i = 0; $i < count($domain); $i++) {
            echo '<div class="video-preview">';
            echo '<div class="thumbnail-row">';
            echo '<video id="my-video" class="video-js thumbnail" controls preload="auto" width="1100" height="500"
data-setup="{}">';
            echo '<source src="' . $domain[$i] . '" type="application/x-mpegURL">';
            echo '</video>';
            echo '</div>';
            echo '<div class="video-info-grid">';
            // Assuming you have static images for channel pic and thumbnail, replace with dynamic values as needed
            echo '<div class="channel-pic">';
            echo '';
            echo '</div>';
            echo '<div class="video-info">';
            echo '<p class="video-title">' . $filenames[$i] . '</p>';
            echo '<p class="video-author">Marques Brownlee</p>';
            echo '<p class="video-stats">3.4M views &#183; 6 months ago</p>';
            echo '</div>';
            echo '</div>';
            echo '</div>';
        }
    ?>
</section>
</main>

<script>
    var player = videojs('my-video');

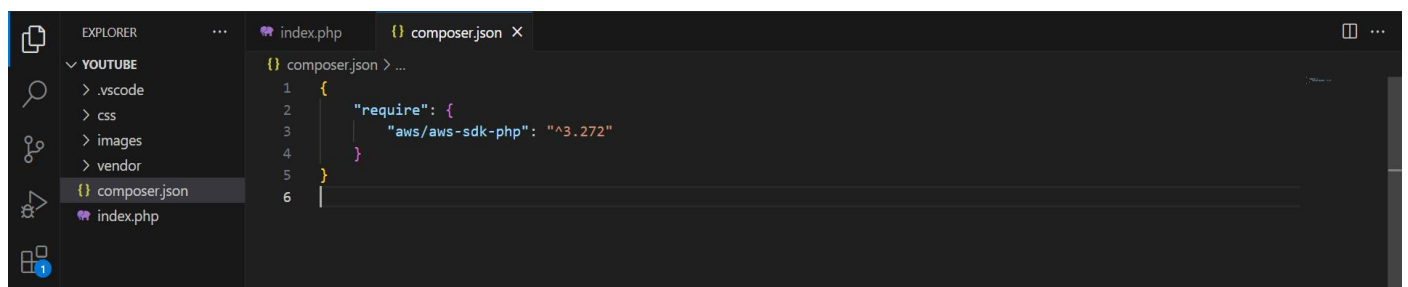
    player.hlsQualitySelector({
        displayCurrentQuality: true,
    });
</script>

</body>
</html>

```

Create a composer.json File:

Create a composer.json file in the root directory of your PHP project. This file specifies the dependencies your project needs. Here's an example of a basic composer.json file:



Composer.json

```

1  {
2      "require": {
3          "aws/aws-sdk-php": "^3.272"
4      }
5  }
6

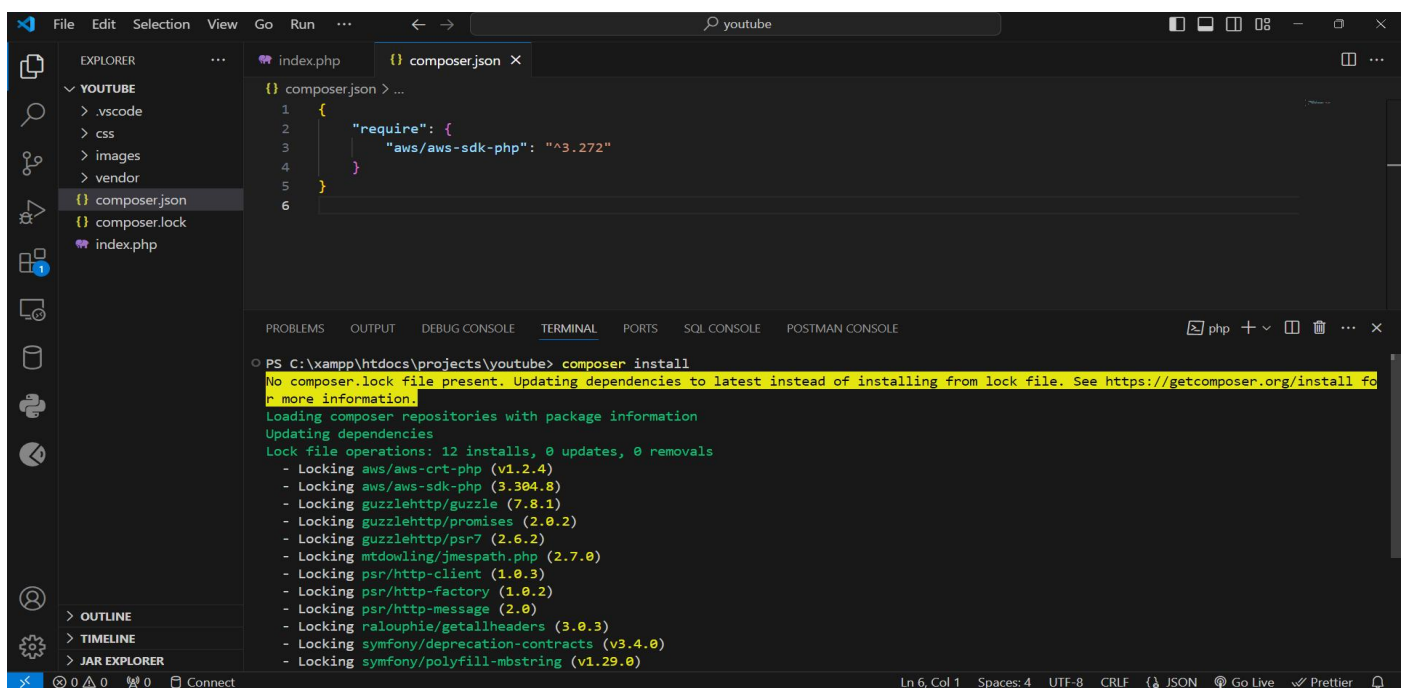
```

The composer.json file should be in project directory otherwise it doesn't satisfy the needs of index.php.

Install Dependencies:

Open a command-line interface (Terminal on macOS/Linux, Command Prompt on Windows) and navigate to the root directory of your project where composer.json is located. Run the following command to install dependencies listed in composer.json .

composer install



```

PS C:\xampp\htdocs\projects\youtube> composer install
No composer.lock file present. Updating dependencies to latest instead of installing from lock file. See https://getcomposer.org/install for more information.
Loading composer repositories with package information
Updating dependencies
Lock file operations: 12 installs, 0 updates, 0 removals
- Locking aws/aws-crt-php (v1.2.4)
- Locking aws/aws-sdk-php (3.304.8)
- Locking guzzlehttp/guzzle (7.8.1)
- Locking guzzlehttp/promises (2.0.2)
- Locking guzzlehttp/psr7 (2.6.2)
- Locking mtdowling/jmespath.php (2.7.0)
- Locking psr/http-client (1.0.3)
- Locking psr/http-factory (1.0.2)
- Locking psr/http-message (2.0)
- Locking ralouphie/getallheaders (3.0.3)
- Locking symfony/deprecation-contracts (v3.4.0)
- Locking symfony/polyfill-mbstring (v1.29.0)

```

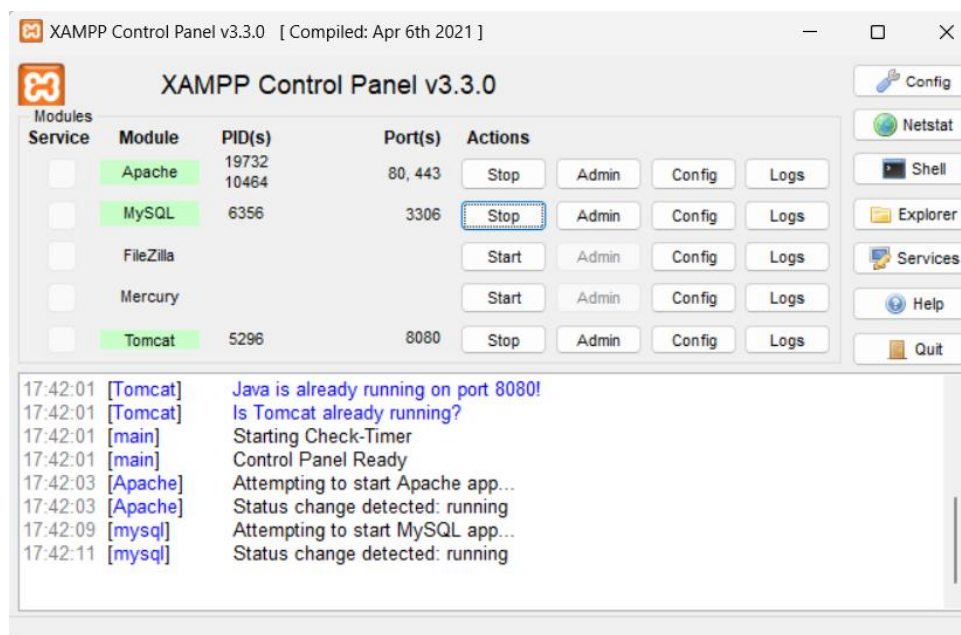
This command reads the composer.json file, resolves dependencies, and installs the required packages into a vendor directory in your project.

TEST THE USER INTERFACE IN LOCAL:

To run a PHP file locally on your machine, you need a web server installed. Since you've already installed XAMPP, which includes Apache (a web server), you can follow these steps to run a PHP file:

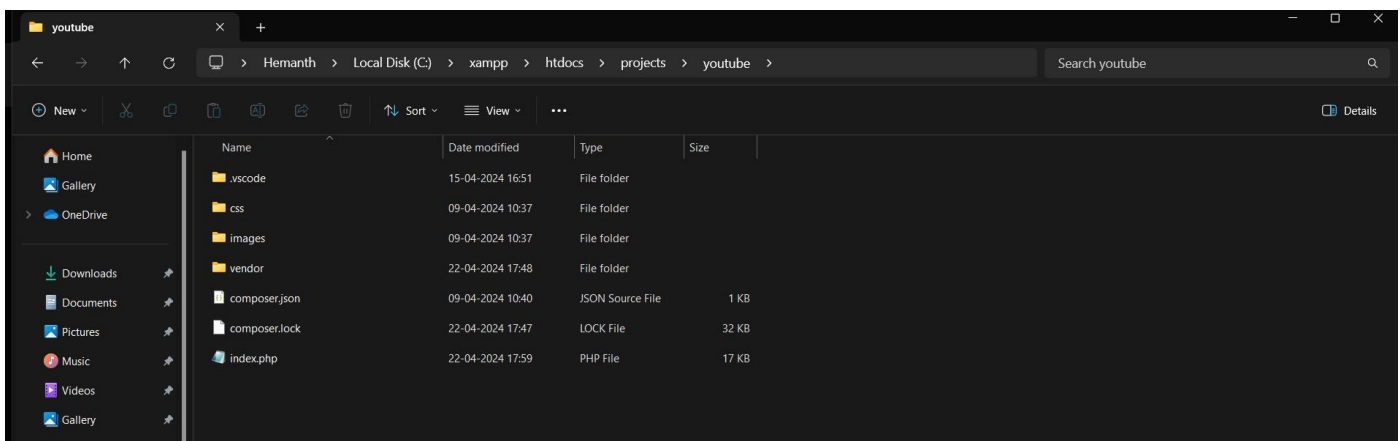
START XAMPP:

- Open the XAMPP Control Panel.
- Start the Apache service by clicking the "Start" button next to "Apache".



PLACE PROJECT DIRECTORY:

- Place your PHP file in the htdocs directory within the XAMPP installation directory. For example, C:\xampp\htdocs on Windows or /Applications/XAMPP/htdocs on macOS.
- **C://xampp/htdocs/projectfolder/index.php**



ACCESS PHP FILE IN WEB BROWSER:

- Open your web browser.
- In the address bar, type `http://localhost/` followed by the name of your PHP file.
- <http://localhost/projectfolder/index.php>.



- Press Enter to load the PHP file. Apache will process the PHP code and display the output in the browser.

VIEW PHP OUTPUT:

- Once you access the PHP file in your web browser, you should see the output generated by the PHP code.
- If there are any errors in the PHP code, Apache will typically display an error message in the browser.

STOPPING XAMPP:

- After you're done testing your PHP file, you can stop the Apache service in the XAMPP Control Panel to free up system resources.
- That's it! You've successfully run a PHP file locally using XAMPP.

CHAPTER V

CONCLUSION

This project has successfully implemented a robust video processing and delivery system leveraging various AWS services. By integrating AWS Lambda functions, Amazon S3, AWS Elemental MediaConvert, AWS Elemental MediaPackage, AWS Secrets Manager, DynamoDB, CloudFront, and other services, we have achieved efficient and scalable video processing, storage, and delivery capabilities.

The project workflow begins with uploading video files to an S3 bucket, triggering Lambda functions to convert the videos into multiple formats using MediaConvert. The converted videos are then stored in another S3 bucket, triggering additional Lambda functions to manage packaging configurations and CloudFront distribution for secure content delivery. Metadata such as filename, endpoints, and domain name are stored in DynamoDB for efficient retrieval and management.

Furthermore, a user interface resembling YouTube has been developed to showcase the processed videos, providing a seamless viewing experience for users. By retrieving endpoints and domain names from DynamoDB, the UI dynamically presents the available video content to users.

Throughout the project, security and compliance have been prioritized, with sensitive information handled securely using AWS Secrets Manager, encryption, and access control mechanisms. Regular testing and monitoring ensure the reliability, performance, and security of the system.

Looking ahead, potential future improvements include optimizing resource utilization, enhancing scalability, and incorporating additional features such as user authentication and content recommendation systems.

In summary, this project demonstrates the effective utilization of AWS services to build a scalable and secure video processing and delivery platform, laying the foundation for future enhancements and innovations.