　　AWS S3にアップロードされたMP4動画を、Lambda×MediaConvertで、HLS配信に最適化したM3U8ファイルに変換しよう！！

**◆【Lambda】**

■[ 解説 ]

|  |
| --- |
| AWS Lambdaは、サーバーレスコンピューティングプラットフォームであり、開発者がコードを実行するためのコンピューティングリソースを管理する必要なく、コードを実行できるサービス。  Lambdaを使用すると、コードをアップロードしてトリガーを設定するだけで、必要に応じて自動的にスケーリングされる実行環境が提供される。  これにより、インフラストラクチャの管理やサーバーのプロビジョニングにかかる手間を省き、開発者はコードの開発に集中できる。 |
| サーバーレス(サーバー管理不要)で処理を実行可能。 |
| トリガーを設定することで、処理を実行。  例：S3にMP4動画がアップロードされたら、M3U8ファイルに変換  ：HTTPアクセスがあったら、webアプリを表示  ：1時間起きに、処理を実行 |

■[ 料金 ]

・公式：https://docs.aws.amazon.com/ja\_jp/whitepapers/latest/how-aws-pricing-works/aws-lambda.html

・割り当てられてたメモリサイズと、実行時間によって決まる。

・

|  |  |  |
| --- | --- | --- |
| リクエスト料金 | 無料 | ・1 か月あたり 100 万件のリクエスト  ・1 か月あたり 400,000 GB 秒のコンピューティング時間 |
| 有料 | ・無料利用枠を使い切った場合、  0.20 USD/1,000,000 件のリクエスト（0.0000002 USD/リクエスト） |
| 期間料金 | 無料 | ・1 か月あたり 400,000 GB秒  ・最大 320 万秒のコンピューティング時間 |
| 有料 | ・無料利用枠を使い切った場合、0.00001667 USD/GB 秒 |

・GB秒

＞Lambdaが実行される時間の単位。

＞コードが実行される際に使用されるコンピューティングリソースの量を表す。

＞実行されたコードが利用するメモリの量（GB）と実行時間（秒）の積で計算。

＞例：Lambda関数が1秒間実行され、メモリが1GB(1024MB)使用されたら、1GB秒が消費されたこととなる。

・本講義では、せいぜい10回程度の簡易的な処理の実行なので、ほぼ料金は発生しない。

**◆【S3】**

■[ 解説 ]

|  |
| --- |
| AWSが提供する安価で可用性が高いクラウドストレージサービス。 |
| S3は自動的にスケーリングされ、大規模なデータセットでも高い耐久性を提供。  データは複数の施設にわたって冗長化され、99.999999999%（11 9's）の耐久性が保証されてる。 |
| 3は使用した分だけ料金が発生する従量課金制のサービス。 |

■[ S3で登場する用語＆概念 ]

|  |  |
| --- | --- |
| **オブジェクト** | ・保存するファイルのこと |
| **バケット** | ・オブジェクトを保存する場所  ・重複しない、ユニークな名前である必要がある |
| **キー** | ・バケットに保存したオブジェクトのURLパス |

■[ 料金 ]

・参考サイト

|  |
| --- |
| https://aws.amazon.com/jp/s3/pricing/ |
| https://docs.aws.amazon.com/ja\_jp/whitepapers/latest/how-aws-pricing-works/amazon-simple-storage-service-amazon-s3.html |
| https://www.c3index.co.jp/blog/blog\_197/ |

・コストの種類

|  |  |
| --- | --- |
| ストレージ料金 | ・S3に保存しているファイルのデータ量に対してかかるコスト  ・0.023USD/GB ＝ 1GB約3円/月 |
| データ転送料金 | ・データ転送料金は、S3内のデータをダウンロードしたり転送したりする際に発生。  ・データ転送時に課金されるが、データをS3にアップロードする際には料金は発生しない。  ・月間、100 GB のインターネットへのデータ転送 (アウト) の無料利用枠あり！！  ・無料利用枠を超えた場合は、東京リージョンの場合、月間で、1GBあたり0.02USD。 |
| リクエスト料金 | ・リクエスト料金は、GETやPUT、POST、などのリクエストの種類応じて発生。   |  |  | | --- | --- | | DELETE、CANCEL | 無料 | | GET、SELECT | 0.00037ドル/1,000リクエスト | | PUT、POST、LIST、COPY | 0.0047ドル/1,000リクエスト | |

**◆【AWS Elemental MediaConvert】**

＊https://tech.curama.jp/entry/2020/07/03/184716

＊https://aws.amazon.com/jp/mediaconvert/pricing/

■[ 解説 ]

|  |
| --- |
| Amazon Web Services（AWS）が提供する動画変換サービス。  S3内のファイルを直接変換可能！！ |
| AWS Elemental MediaConvertは、AWSが提供するクラウドベースのビデオ変換サービス。  このサービスを使用することで、さまざまなビデオフォーマットやコーデック間での変換、ビデオのエンコード(圧縮)や  トランスコード(再エンコード)、その他のビデオ処理タスクを効率的に実行することが可能。 |
| AWS Elemental MediaConvertは、動画の品質を維持しながら異なる形式に変換したり、ビットレートや解像度を調整したりすることが可能。  また、ライブストリーミングやオンデマンドビデオの両方に対応しており、広告挿入や字幕の追加などの機能も提供している。 |

■[ AWS Elemental MediaConvertで登場する用語＆概念 ]

|  |  |
| --- | --- |
| **Job (ジョブ)** | ・MediaConvertで処理を実行する際に必要なもので、  　具体的な処理内容の指示書の様なもの。  ・jobの内容を元に、MediaConvertは処理を実行する。 |
| **Job Template**  **(ジョブテンプレート)** | ・ジョブテンプレートは、ジョブの設定を事前に定義して保存したもの。  ・同じ設定を複数のジョブで使用する必要がある場合や、  定期的に同じ処理を実行する必要がある場合に便利。  ・ジョブテンプレートを使用することで、ジョブの作成が簡素化され、再利用性が向上。 |

　・本講義では、Job Templateを元にjobを生成し、変換処理を実行します。

■[ 料金 ]

|  |
| --- |
| 使用した分だけ料金が発生する従量課金制のサービス。 |
| MediaConvertの料金は、出力ファイルの時間、1分単位での課金。 |
| ベーシック階層とプロフェッショナル階層があり、Pro と記載のあるオプションやメニュー、  パラメータを選択するとプロフェッショナル階層での課金となる。 |
| 2024年2月21日時点で、  東京リージョンで、「動画時間:60分,　サイズ:1080p,　fps:60fps」の動画を「動画時間:60分,　サイズ:720p,　fps:30fps」  の動画に、コーデックAVC(H.264)で変換処理を実行した場合、以下のような料金体系となります  「 60分 × 0.017(米ドル/分) = 1.02(米ドル)」 |

・

・

**ストリーミング と M3U8ファイル について解説**

**◆【ストリーミング】**

■[ 解説 ]

|  |
| --- |
| ストリーミングとは、インターネットを介した動画配信や音楽配信に用いられる配信方式の一種。  データをダウンロードしきってからではなく、徐々にデータを受信しながら同時に随時再生していく。 |
| 「オンデマンド型」と「ライブ型」の2種類がある。   |  |  | | --- | --- | | オンデマンド型 | ・あらかじめサーバーにアップロードされている動画ファイルを、  ユーザーの好きなタイミングで自由に試聴できる配信方式。  ・例：youtube | | ライブ型 | ・リアルタイム配信  ・例：生配信 | |

**◆【M3U8ファイル】**

■[ 解説 ]

|  |
| --- |
| Apple社が開発した、ストリーミング配信に特化した動画のフォーマット。 |
| 拡張子は、「.m3u8」 |
| 元となる動画ファイルを、セグメントと呼ばれる短な動画ファイルに分割し、それらのセグメントを連続して配信することで、ストリーミング再生を実現。  ちなみに、この様な方式**HLS**（**HTTP Live Streaming**）という。 |
| 分割されたセグメントファイルの拡張子は「.ts」で、  通常は、10秒間隔で区切られる。 |
| 60秒の.mp4動画を、.m3u8に変換した場合、  6つの.tsファイルと  1つの.m3u8ファイルが生成される。  ちなみに、.tsファイルは動画ファイルだが、.m3u8ファイルはテキストファイル。 |
| .mp4動画を.m3u8に変換する際、異なるビットレート値の複数パターンのセグメントファイルに分けて、生成することも可能。  そうした場合、ネットワークの状況に基づいて、自動的に最適なビットレートを選択＆切り替えを実行することもできる。。 |

■25秒のxyz.mp4動画を、.m3u8に変換した場合、

3つの.tsファイルと,1つの.m3u8ファイルが生成される。

|  |  |
| --- | --- |
| **テキストファイル.m3u8の中身** | **3つの動画ファイル.tsのファイル名** |
| #EXTM3U  #EXT-X-VERSION:3  #EXT-X-TARGETDURATION:10  #EXT-X-MEDIA-SEQUENCE:1  #EXT-X-PLAYLIST-TYPE:VOD  #EXTINF:10,  xyz\_00001.ts  #EXTINF:10,  xyz\_00002.ts  #EXTINF:5,  xyz\_00003.ts  #EXT-X-ENDLIST | xyz\_00001.ts  xyz\_00002.ts  xyz\_00003.ts |

■25秒のxyz.mp4動画を、highとlowの2クオリティーに分けて、.m3u8に変換した場合、

6つの.tsファイルと,3つの.m3u8ファイルが生成される。

|  |  |
| --- | --- |
| **xyz\_high.m3u8** | **Highクオリティーの.tsファイル群** |
| #EXTM3U  #EXT-X-VERSION:3  #EXT-X-TARGETDURATION:10  #EXT-X-MEDIA-SEQUENCE:1  #EXT-X-PLAYLIST-TYPE:VOD  #EXTINF:10,  xyz\_high\_00001.ts  #EXTINF:10,  xyz\_high\_00002.ts  #EXTINF:5,  xyz\_high\_00003.ts  #EXT-X-ENDLIST | xyz\_high\_00001.ts  xyz\_high\_00002.ts  xyz\_high\_00003.ts |

|  |  |
| --- | --- |
| **xyz\_low.m3u8** | **Lowクオリティーの.tsファイル群** |
| #EXTM3U  #EXT-X-VERSION:3  #EXT-X-TARGETDURATION:10  #EXT-X-MEDIA-SEQUENCE:1  #EXT-X-PLAYLIST-TYPE:VOD  #EXTINF:10,  xyz\_ low \_00001.ts  #EXTINF:10,  xyz\_ low \_00002.ts  #EXTINF:5,  xyz\_ low \_00003.ts  #EXT-X-ENDLIST | xyz\_low\_00001.ts  xyz\_low\_00002.ts  xyz\_low\_00003.ts |

|  |
| --- |
| **xyz\_master.m3u8** |
| #EXTM3U  #EXT-X-VERSION:3  #EXT-X-INDEPENDENT-SEGMENTS  #EXT-X-STREAM-INF:BANDWIDTH=2871436,AVERAGE-BANDWIDTH=1123055,CODECS="avc1.64001f,mp4a.40.2",RESOLUTION=1280x720,FRAME-RATE=30.000  xyz\_high.m3u8  #EXT-X-STREAM-INF:BANDWIDTH=1685683,AVERAGE-BANDWIDTH=732315,CODECS="avc1.64001e,mp4a.40.2",RESOLUTION=640x360,FRAME-RATE=30.000  xyz\_low.m3u8 |
| xyz\_high.m3u8とxyz\_low.m3u8のクオリティーの異なるm3u8ファイルの情報が記載されている。  ブラウザから再生する際は、xyz\_master.m3u8のpathを指定するのみでOK。  ネットワークの状況に基づいて最適なビットレートが選択され、自動的にビットレートの切り替えが実行される。 |

**ビットレート制御方式「CBR,VBR,QVBR」**

**◆【ビットレート制御方式「CBR,VBR,QVBR」】**

■[ 基本解説 ]

|  |  |
| --- | --- |
| CBR | ・一定のビットレートでデータを転送する方式で、固定ビットレートとも言われる。  ・音声や動画のデータを一定のビットレートでエンコード。  これにより、データのビットレートが一定に保つことが可能に。 |
| VBR | ・音声や動画などの圧縮時にビットレートを可変する方式。  ・データの内容の変化に応じて圧縮率を動的に適応させることが可能。  つまり、データの複雑さに応じてビットレートが上下。  これにより、データ全体の平均ビットレートを下げつつ、  必要な部分には高いビットレートを割り当てることができる。 |
| QVBR | 様々なコンテンツやシーンで自動的にビットレートを調整する方式。  これにより、ビットレートはデータの品質に応じて動的に変化します。 |

■[ CBRの採用理由 ]

|  |
| --- |
| 単純にデータ量を抑えたいなら、VBRやQVBRがお勧め。    一方のCBRは、変換時にパディング(余分なデータ)を挿入することでデータ量を一定に保つたことを実現しているため、  VBRやQVBRと比べてデータ量が大きくなりがち。    だが、HLS配信の場合はCBRがベストプラクティスとされている。    その理由は、CBRのようにデータ量を一定に保つことは、HLS配信において、バッファリング(停止や遅延)を防ぐことにつながるためである。ネットワークの帯域幅(水が流れるホースの様なイメージでOK)に対して、常に(ある程度)一定のサイズのデータが通過していくこととなるので、スムーズかつシームレス(継ぎ目のない)なストリーミング体験を提供できる。    逆に、VBRやQVBRでMP4動画をM3U8に変換した場合、総データ量は抑えられる一方、各セグメント(コマ切れの動画ファイル)ごとでかなりのデータ量の差が生じる。この差分が原因で、ネットワークの帯域幅によっては、バッファリング(停止や遅延)を引き起こす場合があるとされている。 |

**AWS IAM(Identity and Access Management)**

**◆【AWS IAM(Identity and Access Management)とは?】**

■[ 解説 ]

・AWS内のどの「サービス」に対しての、どの様な「アクション」を許可/拒否するかを、設定するためのサービス。

■[ IAMの3つの重要概念 ]

・IAM Policy

・IAM User

・IAM Role

**◆【IAM Policy】**

■[ 解説 ]

|  |
| --- |
| AWS内のどの「サービス」に対しての、どの様な「アクション」を許可/拒否するか の設定データ。 |
| この「IAM Policy」は  ユーザに対して権限を付与したい際は、「IAM User」 に  AWSサービス(S3,Lambda)に対して権限を付与したい際は、「IAM Role」に  付与する形で使用する。 |

■[ 「IAM Policy」は以下のようなJson形式のデータ ]

・以下は、S3に対しての読み取り操作の権限を付与するIAM Policyの例

|  |
| --- |
| {  "Version": "2012-10-17",  "Statement": [  {  "Effect": "Allow",  "Action": [  "s3:Get\*",  "s3:List\*",  "s3:Describe\*",  "s3-object-lambda:Get\*",  "s3-object-lambda:List\*"  ],  "Resource": "\*"  }  ]  } |

・IAM Policyは自分で記述して作成することもできるが、事前にAWS側で用意されているのものを使用することが大半。

**◆【IAM User】**

■[ 解説 ]

|  |
| --- |
| 開発者などの人間が AWS にログインして操作するための、AWSユーザのこと。 |
| AWSアカウント作成時に、作られるユーザは、ルートユーザーといわれる特別なユーザーで  AWSの全てのサービスを操作可能な最も強い権限が付与されている。 |
| ルートユーザー以外にも、複数アカウントを作成可能で、  各アカウントごとにアクセス権限データのIAM Policyをアタッチすることで、自在に権限を付与することができる |

**◆【IAM Role】**

■[ 解説 ]

|  |
| --- |
| AWSのLambdaやMediaConvertなどの各サービスに対して、  アクセス権限データのIAM Policyをアタッチする際に用いる。 |
| AWSのサービスに対して、直接IAM Policyをアタッチすることはできない。  　→IAM Roleに対して、アクセス権限データのIAM Policyをアタッチ  →IAM Roleを、AWSのサービスに対してアタッチ。  こうすることで、AWSのサービスに対して、権限を付与することができる。 |
| 例：  →RoleForMediaConvertという名前でIAM Roleを作成し、  S3に対しての操作権限を持ったIAM Policyをアタッチ  →IAM RoleのRoleForMediaConvertを、MediaConvertにアタッチ  →MediaConvertからS3の操作が可能に！！ |

**Lambda関数のPyhtonコード**

**◆【**

**import os**

**import urllib.parse**

**import boto3**

**def lambda\_handler(event, context):**

#Lambda関数がS3イベントをトリガーとして呼び出された際に、そのイベントからバケット名を抽出

**bucket = event['Records'][0]['s3']['bucket']['name']**

#Lambda関数がS3イベントをトリガーとして呼び出された際に、

#そのイベントからオブジェクト(ファイル)のkey名を取得～URLデコード

**key = urllib.parse.unquote\_plus(event['Records'][0]['s3']['object']['key'], encoding='utf-8')**

#保存先のディレクトリ名の値で使用 #例「mp4/abc/example.mp4」

**addDirName = key[:-4]** #拡張子「.mp4」除外 #例「mp4/abc/example」

**addDirName = addDirName[4:]**  #このLambda関数のトリガーには、S3のディレクトリ「mp4/」を設定。

#その「/mp4」を除外 #例「abc/example」

**addDirName = "\_".join(addDirName.split('/')) #例「abc\_example」**

#□MediaConvertジョブを実行する関数「make\_settings」に引数を渡して呼び出し

#　この変数「settings」は、client.create\_jobのパラメータで使用。

**settings = make\_settings(bucket, key, addDirName)**

#ジョブに追加する任意のメタデータ。今回は、'JobCreatedBy': 'videoConvertSample'というユーザーメタデータを指定。

**user\_metadata = {**

**'JobCreatedBy': 'videoConvertSample',**

**}**

#Boto3を使用してAWS Elemental MediaConvertサービスに対するクライアント(APIクライアント)を作成

**client = boto3.client('mediaconvert', endpoint\_url = "XXX")**

#AWS Elemental MediaConvertサービスに対して新しいメディア変換ジョブを作成するためのリクエストを送信

**result = client.create\_job(**

#メディア変換ジョブを実行するためのIAMロールのARN

**Role = "XXX",**

#作成するメディア変換ジョブに使用するジョブテンプレート。\*事前に作成しておく

**JobTemplate = "mp4ToHls",**

#「make\_settings(bucket, key)」で作成したディア変換ジョブの詳細な設定を含む情報を指定

Settings=settings,

#ジョブに追加する任意のメタデータ。今回は、'JobCreatedBy': 'videoConvertSample'というユーザーメタデータ。

**UserMetadata=user\_metadata,**

**)**

**def make\_settings(bucket, key, addDirName):**

#拡張子を取り除いた値

**basename = os.path.basename(key).split('.')[0]**

**return \**

**{**

**"Inputs": [**

**{**

**"FileInput": f"s3://{bucket}/{key}",**#加工対象のpath

**"AudioSelectors": {**

**"AudioSelector1": {**

#オーディオストリームの開始位置を秒単位で指定。デフォルトは0。

**"Offset": 0,**

#デフォルトのオーディオストリームを選択する方法を指定。

#"DEFAULT"を指定すると、入力コンテナがデフォルトのオーディオトラックを定義している場合、

#そのトラックが選択される。

**"DefaultSelection": "DEFAULT",**

#プログラム番号を指定。

#これは、複数のプログラム(例えば、マルチプログラムトランスポートストリームなど)が含まれる入力を

#処理する際に使用。この例では、最初のプログラム(1)が選択される。

**"ProgramSelection": 1**

**}**

**},**

**}**

**],**

#出力グループをリストで定義。複数指定可能。

**"OutputGroups": [**

**{**

#出力グループに名を任意で命名。

**"Name": "Apple HLS",**

#出力グループの設定を指定

**"OutputGroupSettings": {**

#出力グループのタイプを指定。今回は、。HLS形式の出力を行うために、HLS\_GROUP\_SETTINGSを指定

**"Type": "HLS\_GROUP\_SETTINGS",**

#出力するHLS形式に関する詳細設定をオブジェクト形式で指定。

**"HlsGroupSettings": {**

#HLS出力ファイルの保存先となるS3バケット内のディレクトリを指定

**"Destination": f"s3://{bucket}/m3u8/{addDirName}/",**

**},**

**},**

#このグループ内の出力設定をリストで指定。ここでは1つの出力設定を定義。

**"Outputs": [**

**{**

**"VideoDescription": {**

**"Width": 720, "Height": 404,**

**},**

#出力オーディオストリームの設定。「Inputs>AudioSelectors>AudioSelector1」の設定を適応。

**"AudioDescriptions": [**

**{**

**"AudioSourceName": "AudioSelector1"**

**}**

**],**

**},**

**],**

**},**

**],**

**}**

**】**

■参考URL

・https://docs.aws.amazon.com/ja\_jp/mediaconvert/latest/ug/example-job-settings.html

■lambdaで設定したS3に対してのトリガーは、

「Amazon S3 > バケット > バケット名> プロパティー > イベント通知」箇所で確認可能。

■[ import boto3」]

boto3は、AWSをPythonから操作するためのライブラリ。

boto3を使うと、S3やEC2などのAWSのサービスをPythonのコードで簡単に扱うことが可能。

例えば、S3にファイルをアップロードしたり、EC2のインスタンスを起動したり、停止したりするなどなど。

■[ bucket = event['Records'][0]['s3']['bucket']['name'] ]

このコードの該当部分は、AWS Lambda関数がS3イベントトリガーとして呼び出された際に、そのイベントからバケット名を抽出している。

具体的には、Lambda関数がイベントを受け取ると、そのイベントオブジェクト内にはS3イベントに関する情報が含まれている。

その中で、event['Records'][0]['s3']['bucket']['name']は、S3バケットの名前を取得している。

■[ key = urllib.parse.unquote\_plus(event['Records'][0]['s3']['object']['key'], encoding='utf-8') ]

このコードの該当部分は、Lambda関数がS3イベントトリガーとして呼び出された際に、

そのイベントからS3オブジェクトのキー(ファイルのパスや名前)を抽出している。

具体的には、event['Records'][0]['s3']['object']['key']からキーを取得し、それをurllib.parse.unquote\_plusを使用してURLデコード処理を実行。

■[ settings = make\_settings(bucket, key, addDirName) ]

MediaConvertジョブを実行する関数「make\_settings」に引数を渡して呼び出している。

この変数「settings」は、client.create\_jobのパラメータで使用。

■[ client = boto3.client('mediaconvert', endpoint\_url = "https://xxxxxx.mediaconvert.ap-northeast-1.amazonaws.com") ]

AWS SDK for Python(Boto3)を使用してAWS Elemental MediaConvertサービスに対するAPIクライアントを作成。

この行の各要素の解説：

・boto3.client('AWSサービス名','対象のAWSサービスのエンドポイント'):

＞Boto3ライブラリを使用してAWSサービスに対するクライアントを作成する関数。

・'mediaconvert':

＞AWSサービス名

＞作成するクライアントが対象とするAWSサービス名。

この場合、AWS Elemental MediaConvertサービスに対するクライアントを作成。

・endpoint\_url="https://xxxxxx.mediaconvert.ap-northeast-1.amazonaws.com":

＞対象のAWSサービスのエンドポイント

、、、

これにより、client変数はAWS Elemental MediaConvertサービスに対するクライアントとして機能し、後続のコードでMediaConvertサービスへのリクエストを行うために使用可能となる。

このクライアントを使用すると、MediaConvertサービスのAPIを呼び出してメディア変換ジョブを作成するなどの操作が可能。

■[

settings = make\_settings(bucket, key, addDirName)

user\_metadata = {

'JobCreatedBy': 'videoConvertSample',

}

result = client.create\_job(

Role = "RoleForMediaConvertのARN",

JobTemplate = "mp4ToHls",

Settings=settings,

UserMetadata=user\_metadata,

)

]

AWS Elemental MediaConvertサービスに対して新しいメディア変換ジョブを作成するためのリクエストを送信。

各要素の解説：

・client：

＞「boto3.client('AWSサービス名','対象のAWSサービスのエンドポイント'): 」で生成した、

MediaConvertサービスのclientインスタンス

・client.create\_job:

＞client変数(MediaConvertサービスへのクライアント)を使用して、

新しいメディア変換ジョブを作成するためのメソッドを呼び出し。

・Role:

＞メディア変換ジョブを実行するためのIAMロールのARN(Amazon Resource Name)を指定。

IAMロールは、MediaConvertサービスがアクセス権を持つための権限を提供。

今回は、IAMロールのARNが指定。

・JobTemplate:

＞作成するメディア変換ジョブに使用するジョブテンプレートの名前を指定。

ジョブテンプレートは、変換ジョブの設定やプリセットを定義するために使用される。

今回は、"mp4ToHLS"というジョブテンプレートが指定。

＞予め、ジョブテンプレートを作成しておく必要がある。

・Settings:

・settings = make\_settings(bucket, key, addDirName)

＞MediaConvertジョブを実行する関数「make\_settings」に引数を渡して呼び出している。

この変数「settings」を、client.create\_jobのsettingsパラメータに指定することで、

make\_settings関数内のreturn内容に応じてMediaConvertジョブの設定情報を生成

＞ディア変換ジョブの詳細な設定を含む情報を指定。

・UserMetadata:

・user\_metadata = {'JobCreatedBy': 'videoConvertSample'}

＞ユーザーが任意のメタデータをジョブに追加できる領域。

今回は、'JobCreatedBy': 'videoConvertSample'というユーザーメタデータが追加。

これは、ジョブを作成したユーザーに関する情報などを含めるために使用できる。

メディア変換ジョブの作成リクエストが成功すると、result変数には作成されたジョブに関する情報が代入されることとなる。この情報には、ジョブのID、ステータス、作成日時などが含まれる。この情報を使用して、後続の処理やジョブの監視をすることが可能。

■[

"OutputGroups": [

{

"Name": "Apple HLS",

"OutputGroupSettings": {

"Type": "HLS\_GROUP\_SETTINGS",

"HlsGroupSettings": { "Destination": f"s3://{bucket}/output/" }

},

"Outputs": [

{

"VideoDescription": { "Width": 640, "Height": 360 }

},

],

},

],

]

MediaConvertジョブの出力設定を定義。具体的には、HLS（HTTP Live Streaming）グループを作成し、

その中に1つの出力を指定している。

各要素の解説：

・"OutputGroups":[---]

＞出力グループをリストで定義。1つのジョブで複数の出力グループを定義することもできるが、

今回は1つのグループのみを定義。

・"Name": "Apple HLS"

＞出力グループに名前を指定。この名前は任意で、ジョブの設定を理解しやすくするために使用する。

・"OutputGroupSettings":{---}

＞出力グループの設定を指定。

・"Type": "HLS\_GROUP\_SETTINGS"

＞この出力グループのタイプを指定。HLS形式の出力を行うために、HLS\_GROUP\_SETTINGSが指定されている。

・"HlsGroupSettings":{---}

＞ 出力するHLS形式に関する詳細設定をオブジェクト形式で指定。

・"Destination": f"s3://{bucket}/output/"

＞ HLS出力ファイルの保存先となるS3バケット内のディレクトリを指定。

・"Outputs":[---]

＞: このグループ内の出力設定をリストで指定。ここでは1つの出力設定を定義。

・"VideoDescription":{---}

＞出力ファイルに関する詳細設定をオブジェクト形式で指定。

・"Width": 640

＞ 出力ビデオの幅を指定。この例では640ピクセル。

・"Height": 360

＞出力ビデオの高さを指定。この例では360ピクセル。

このように設定された出力グループは、MediaConvertジョブが実行された際に、指定されたHLS形式の設定に従ってビデオを変換し、指定されたS3バケットとディレクトリに保存します。

　　投稿予定の講座

**◆【**

**S3 × Lambda × MediaConvert を用いて、**

**mp4動画 を HLSストリーミング配信可能な M3U8 ファイルに変換しよう！！**

**】**

**◆【Nest.js×Next.js!!フルスタックでブログ開発!!】**

？バックエンドもNext.jsにして、独自ドメイン紐づけ～Amplifyで公開に変更するかも？

□[ 機能 ]

|  |
| --- |
| ログイン時のメール認証 |
| 携帯電話番号を用いたSMS認証 |

□[ 画像保存先 ]

|  |
| --- |
| AWS-S3 |

**◆【HLSでの動画配信!!「.m3u8」ファイルを徹底的解説。】**

□フロントエンド ：Next.js

□バックエンド ：Nest.js

□機能：

|  |
| --- |
| 自作のvideoコンポーネントを作成。  シークバーにマウスホバー時、その時点の映像を表示。 |
| ffmpegをバックエンドで起動し、mp4動画を、m3u8に変換 |

**◆【仮想通貨Ethereumを用いた報酬支払機能を用いて、ASP(広告サイト)を作成しよう！！】**

|  |  |
| --- | --- |
| **フロントエンド** | Next.js |
| **バックエンド** | Nest.js |
| **機能** | Ethereumを用いた報酬支払機能 |
| amazon,rakuten,dmmいずれかのAPIを採用し、広告を表示し、  クリックor動画再生のタイミングで報酬加算。 |

**◆【仮想通貨Ethereumを用いて決済システムを構築し、コンテンツ販売アプリを作成】**

|  |  |
| --- | --- |
| **フロントエンド** | Next.js |
| **バックエンド** | Next.js |
| **機能** | Ethereumを用いた決済システムの構築 |

**◆【フルスタックで開発！！動画投稿プラットフォーム】**

＊最後＊

|  |  |
| --- | --- |
| **フロントエンド** | Next.js |
| **バックエンド** | Next.js |
| **データベース** | RDS |
| **インフラ** | ECS Fargate??予定 |
| **動画ファイルの形式** | HLS(.m3u8) |
| **動画の保存先** | AWS-S3 |
| **セキュリティー** | XSS,CSRF,Brute-force,DOS,DDOS |
| サーバーサイドcookie, csrfToken, rateLimit  WAF?? , aws shield advanced(返金保証あり)?? |