一般社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
IEICE Technical Report
LOIS2022-63(2023-03)

Moodle 用動画配信システムの動画管理サイトを対象としたオンプレミス・サーバーのクラウド化の検討

田中　香次†　　檜垣　泰彦‡

† 千葉大学工学部総合工学科都市環境システムコース　〒263-8522　千葉市稲毛区弥生町 1-33
‡ 千葉大学アカデミック・リンク・センター　〒263-8522　千葉市稲毛区弥生町 1-33
E-mail:　† higaki.yasuhiko@faculty.chiba-u.jp

あらまし　千葉大学では教育機関がオンライン授業をする上で必要とされる条件を定義し，それらを満たすために必要な動画パッケージを作成するシステムを開発している．このオンプレミスのサーバーで動作している動画管理サイトをクラウド環境へ移行するための検討を行った．具体的には現システムの Perl/CGI による MPA を Node.js と Vue.js を使用することで SPA としサーバー負荷を軽減した．また，クラウドが分散環境であることから，ストレージにオブジェクトストレージS3を使用して設計を行うことで,クラウド上デプロイが可能であることを示した．そして，オートスケーリング，ロードバランサーを採用することで，サーバー負荷の増加にも対応できるものとした．

キーワード　HLS，AWS，Docker，S3，クラウドフロント，FFMPEG，SCORM，Moodle，メディアクラス

Conversion from on-premise servers to cloud computing for a video-management system of a Moodle video-streaming system

Koji TANAKA†　　Higaki YASUHIKO‡

† Faculty of Engineering, , Chiba University　1-33 Yayoi-cho, Inage-ku, Chiba-shi, 263-8522 Japan
‡ Chiba University Academic Link Center　1-33 Yayoi-cho, Inage-ku, Chiba-shi, 263-8522 Japan
E-mail:　† higaki.yasuhiko@faculty.chiba-u.jp

Abstract　Chiba University has defined the requirements for online classes and created a video management system to fulfill them. We have studied the possibility of migrating the video management site running on an on-premise server to the cloud environment. The current system's Perl/CGI-based MPA was replaced with SPA using Node.js and Vue.js to reduce the server load. In addition, since the cloud is a distributed environment, we showed that it is possible to deploy the system on the cloud by designing the system using S3 object storage. The system can also cope with the increase of server load by adopting auto-scaling and load balancing.

Keywords　HLS，AWS，Docker，S3，CloudFront，FFMPEG，SCORM，Moodle，Media Class

1. 序論

1.1 背景

Moodle 用動画配信システムの構成について述べた文献[1]において，動画管理システムは Perl/CGI で実装された MPA(Multiple Page Application)のオンプレミスのサーバーとして構成している．学内のオンプレミスのサーバーの運用は，その維持管理にかかる費用な管理の手間から，クラウドサービスへの移行が検討されている．本研究では，この Moodle 用動画配信システムの動画管理サイトを対象として，オンプレミスのサーバーのクラウドへの移行について検討を行った．

1.2 目的

本研究では，文献[1]の動画管理サイトについて，代表的なクラウドサービスの一つである AWS(Amazon Web Service)への実装について検討すると同時に，その機能性の向上を図る．即ち，本研究の目的は以下の二点である．

1. 文献[1]と比較しながら再設計を行い，動画管理サイトの機能向上を図る
2. クラウド化の検討を行う．即ち，クラウド環境で実装可能な設計とし，AWS でデプロイするためのインフラ設計を行う

2. 必要要件

文献[1]において，動画管理サイトが満たすべき必要機能を定義する.本システムで実装される機能は以下の七つであり，図 1 に全体構成を示す.

a) 認証機能
b) MP4 ファイルのセグメント化
c) 動画ファイル管理
d) ビデオファイル検索機能
e) セグメントファイルのプレビュー
f) SCORM パッケージの生成
g) クラウド環境での実装

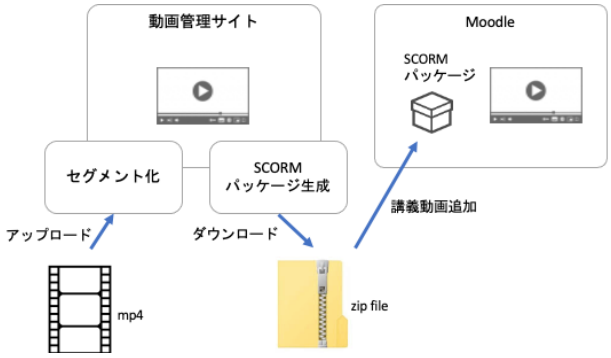

図 1 動画配信システムの動画管理サイトの構成

3. アプリケーション設計

3.1 認証機能実装方法の検討

本システムにはユーザー認証が必要であり，そのためにはセッション管理を行わなければならない.セッションの管理方法には 1.Session ID を Cookie に格納する方法，2.JWT(JSON Web Token)を Cookie に格納する方法，3.JWT をローカルストレージに保存する方法，4.Session ID とトークンを利用する方法がある． XSS の対策を行った上で，JWT の有効期限を設定し Cookie の設定で安全性を高めることで，十分なセキュリティーを確保できると考えられる． また， Session ID を使用する方法の場合，クラウドなどの分散環境でデータベース設計の複雑性が高まるため，3.JWT を Cookie に格納する方法を採用する． このシステムを図 2， 3 に示す.

JWT はユーザーの ID を保持しており， トークンの改ざんの有無のみをチェックする． また， JWT トークンに有効期限を設けることでトークン漏えい時の不正利用の被害を最小限とする．JWT は Cookie に格納され HTTP only フラグを使用することでブラウザの JavaScript 上で改竄されることを防ぎ，HTTPS 通信でのみ Cookie を送信するよう実装した.

図 2 ログイン認証

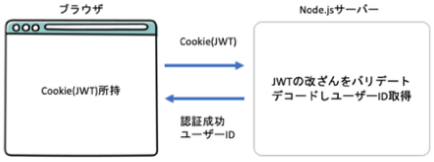

図 3 ログイン後認証

3.2 セグメントファイル化

本システムでは動画配信方法をコンテンツ保護の観点からストリーミング方式としており，HTTP Live Streaming (HLS)を使用している[2][3]． ストリーミング再生を実現させるため， ffmpeg[4]で Mp4 形式の動画を ts セグメントファイル化し，そのメタファイルである M3U8 ファイル作成する．これには libx264 ライブラリを用いる．M3U8 プレイリストファイルとセグメントファイルによるストリーミング配信の構成を図 4 に示す．プレイリストを参照し，各セグメントを取得して動画を再生することが可能な構成となっている.

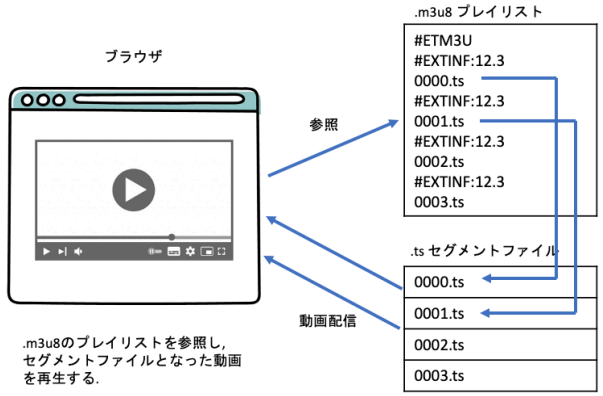

図 4 ストリーミング再生

また，本システムで変換した動画を後日使用したい場合のユースケースために動画を管理する機能を持っている．ffmpeg では動画のセグメント化に加えて，サムネイルの作成も行なっており，MP4 ファイルからスクリーンショットを取得する.

動画処理は ffmpeg によって行われ，並列処理により，処理を複数スレッドに分散している.

3.3 動画の保存

セグメント化された動画は安全に保存されバックアップが取られなければならない．動画の保存にはセグメント化するサーバーに保存する方法やクラウドのオブジェクトストレージに保存する方法，マウントシステムを利用する方法などが考えられる．そこでそれらの手法を比較検討した．

オンプレミスの場合ローカルのファイルに保存することもできるが，クラウド環境ではそれが難しくなる．特に今回の開発ではオートスケーリングを実装するためコンテナ内のファイルのみに保存することは不可能である．クラウド上での画像や動画の保存方法として最も一般的に用いられるシステムはオブジェクトストレージである．AWS で導入されているオブジェクトストレージは S3[5]であり，そのオブジェクトの保存は 99.999999999%安全と言われ，データを紛失する可能性は極めて低い．よって，ファイルサーバーでの保存よりも遥かに耐久性が高いと言える．また，使用に際して発生するコストが低く，スケーラビリティも高い．

AWS では動画の保存用サービスとして S3 の他に，EBS(Elastic Block Store)なども用意されておりファイルをマウントすることが可能である．しかし，これは EC2 インスタンスのストレージボリュームをアタッチするために使われる．本開発では実行環境に EC2 では無く Fargate を選択し，Fargate はマウントをサポートしていないため EBS は使用できない．

本システムの環境がクラウドの分散環境であること，実行環境の Fargate がマウントに対応していないこと，S3 で保存の安全性と費用の節約が期待できることから，本システムのストレージには S3 が最も適当であると考えられる．

3.4 管理・プレビュー・ダウンロード

動画ファイルの管理機能概要を図 5 に示す．動画ファイルを S3 に保管することに加えて，その動画の情報をデータベースで管理する必要がある．データベースの選定には RDB と NoSQL があるがあり，既存のシステムでは RDB を使用している．しかし，RDB のみに実装されている機能を使用しない上，開発で SQL インジェクションの対策を除外でき，実装時に検索速度が向上することからドキュメントベース NoSQL である MongoDB を使用する．

ブラウザ上では動画の検索なども可能でプレビューは Video. js を使用して行い，選択した動画で SCORM パッケージを生成し，その ZIP ファイルをダウンロードをすることが可能である．

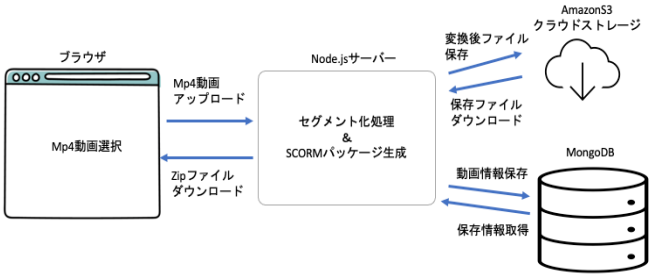

図 5　動画ファイルの管理機能概要

4. インフラ設計

4.1 Docker コンテナの AWS 上実装方法の検討

Docker コンテナはアプリケーションを開発・実行するための仮想環境を構築するシステムであり，仮想マシンに比べ軽量である．また，クラウドでの実装が容易となるためこれを採用する．図 6 は AWS のデプロイ概要である．コンテナの静的ファイルである Docker Image を AWS の ECR[6]に登録し，それをコンテナオーケストレーションサービスである ECS[7]で実行することでサーバーサイドアプリケーションをリリースする．

ECR ではイメージのバージョンを管理することが可能で，アップデートの影響で問題が起こった場合に迅速に前のバージョンに戻すことができる．また，Docker Hub に登録してある Docker Image はアップロードせずともイメージを指定さえすれば ECR が Docker Hub からダウンロードする．

ECS の中にはクラスター，サービス，タスクの三つの構成要素があり，それを設定することで分散環境の作成が可能となる．

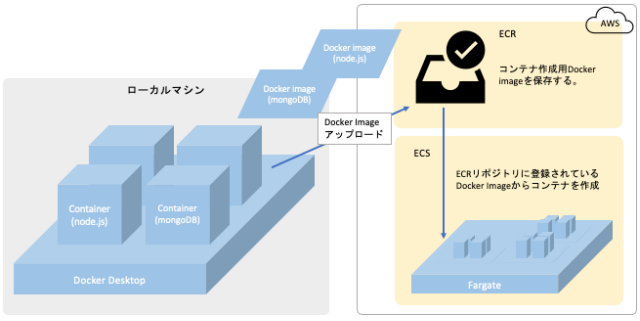

図 6　AWS での Docker コンテナ実行

4.2 実行環境検討

AWS での実行環境には EC2[8], Fargate[9]が候補としてあげられる．

EC2 はクラウド上の仮想マシンであり，CPU やメモリがあらかじめ割り当てられる．そして費用はリクエスト量に関係なく決定される．対照的に Fargate はサーバーレスコンピューティングを行い，動的に CPU や

メモリが割り当てられ，システム使用量に応じて費用が発生する．しかし，このサーバーレスの設計により初回リクエスト時に CPU やメモリーなどを用意する必要があり，この時にシステムでの処理開始までのラグが生じてしまう可能性がある．また，設計方法や使用状況によっては Fargate よりも EC2 の方がコストが低くなることもある．

教育機関では期間によってシステム使用頻度が増減することが予想され，Fargate を適切に設定して使用すれば，一定の処理能力を出すことが可能な状態を維持しながら，全体としてコストを抑えることができる．Fargate を使用した場合のコストが必要以上に高くなってしまう理由としては vCPU の割り当て過ぎが考えられる． vCPU の割り当てを最小限に抑えながら，リクエストが増加したときは横方向スケーリングで対応することで vCPU の割り当てすぎとなる問題を解決できる． このため必要な vCPU を調べる必要があるが，これは CloudWatch を使用することで可能となる．適切な設定により，Fargate の方がコストが低くなることから，EC2 ではなく Fargate を採用する．

4.3 タスク定義

ローカル環境ではコンテナ間の関係性，ネットワーク，ボリュームの設定を行う docker-compose.yml ファイルを作成し開発をした．クラウド環境ではこれに加えて CPU やメモリの設定も行う必要があり，その変数の定義をタスク定義で行う．

本システムのタスク定義で設定するパラメータは以下の五つである．

- 各コンテナで使用する Docker イメージ
- 各タスクでの CPU とメモリの量
- タスクで使用するログの設定
- 必須タスクかどうかの設定
- コンテナが終了した場合のタスクの再起動設定

4.4 vCPU 設定

本システムでは動画のセグメント化処理で最もサーバーの負荷が重く，CPU を使用する．クラウドでは vCPU(virtual CPU)と呼ばれる仮想的な CPU を設定できる．この vCPU は最低て 0.5 vCPU とできるがこれでは講義に必要な 90 分の動画をセグメント化するために非常に長い時間を要してしまう．この状態でテストを行ったところ， AWS の vCPU 使用状況は 100%となりメモリー使用量も増加した． CPU が 100%使用されていることから，この状況で並行処理した場合に処理速度が遅くなることが予想される．そのため，CPU が専有されている状態で次のリクエストを受け取った場合はタスクをスケーリングすることが必要である．

AWS では高 vCPU を設定することができるが，費用が高額となるため使用者の状況に応じて，必要最小限を検討する必要がある.この CPU の使用量は動画の画面解像度に大きく依存する．そこで解像度によってどの程度 CPU を使用しているかを調査した．

表 1. 動画解像度と CPU 使用量

動画解像度	CPU (1 コア=100%)
4096×2160	536%
3840×2160	525%
1920×1080	418%
720×480	420%

表 1 から分かるようにノート PC 内蔵カメラで撮影されたもの(1280×720 程度)で 4vCPU 程度必要となり，Node.js の QUEUE のコンカレンシーを 2 に設定，ffmpeg で並列処理可能としていることから，最低で 8vCPU が搭載されるべきであると予想される．しかし，クラウドの vCPU 数は他の処理の影響も受けるためクラウドに実装後，変換される動画の解像度を決定し，CloudWatch で確認して決定されるべきである．

4.5 AWS でのクライアントサイド実装

本システムでは操作性の向上， サーバー負荷の軽減の理由から現在使用されている MPA では無く，Vue.js による SPA(Single Page Application)を採用している． SPA は MPA のようにページの切り替えごとに HTML を読み込まず，初めの訪問で一枚の HTML をロードする．この仕組みによってクライアントサイドが独立しているため，そのインフラ設計を行う必要がある． 図 7 に示すように S3 からの静的ホスティング，ロード速度を上げるための CloudFront (CDN)を使用する． 加えて， 公的機関が使用することから Route 53 (DNS)でドメイン名を設定する．

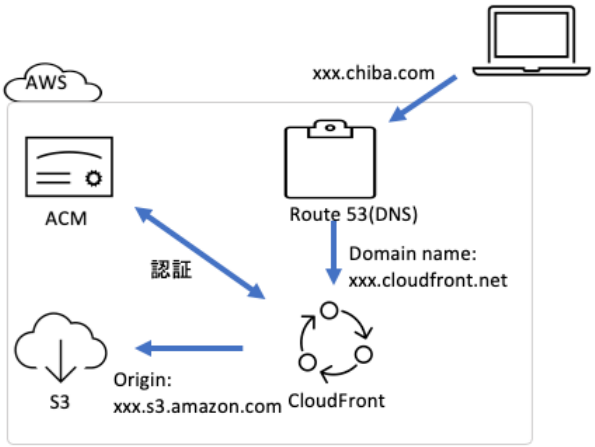

図 7　SPA デプロイ

4.6 オートスケーリング・ロードバランサー実装

サーバーへのリクエスト数が増加したときに，新しいタスクを起動して対応できることはクラウドの利点の一つである．前項目で示した通り CPU の使用量は動画の解像度と変換する動画の並列処理数に依存する．複数リクエストがあった場合に用意されている vCPU 以上が必要となり処理速度の低下を防ぐため，セグメント化処理をする関数を QUEUE に格納し，設定された数の並行処理を行う．この時メモリ使用量が増加する．加えて解像度が高い動画を送られた場合にも対応する必要があるため，CPU とメモリの使用状況の管理を Cloud Watch で行うことで，オートスケーリングのための使用状況管理を行う．ロードバランサーは複数タスクが起動した場合にそれぞれのタスクに負荷を分散させることができるよう使用される．そのためクライアントサイドからのリクエストは図 8 のように ELB(Elastic Load Balancing)に送られ，それらのリクエストが各タスクに送られる．オートスケーリングするタスクは動画変換処理サーバーであり，そのタスクが増加する．一方でデータベースサーバーは増加させた場合に情報の整合性が取れなくなるためオートスケーリングしない．

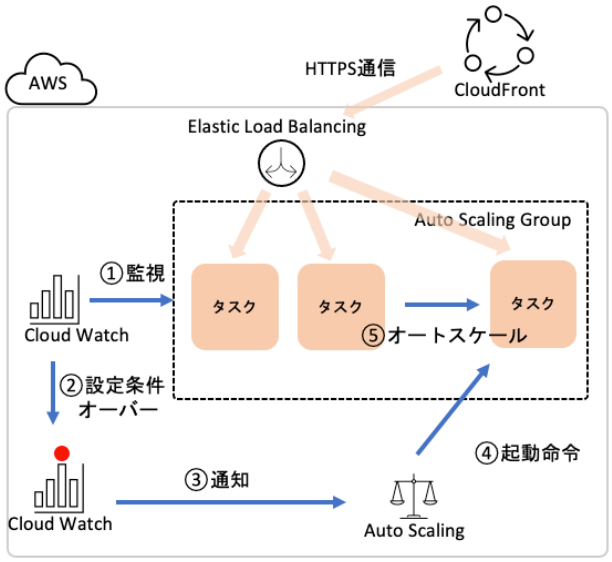

図 8 オートスケーリングとロードバランサー

4.7 セキュリティーグループ設計

AWS Fargate では VPC(virtual private cloud)とサブネットの設定が必須となっている．VPC は AWS のユーザー情報に紐付けて作成することが可能な論理的に分離された仮想ネットワークである．ユーザーに仮想プライベートクラウドを提供することでネットワークを隔離することが可能になる．その中で各サービス用に CIDR によって指定された幅の IP アドレスをサブネットと呼び，その中にタスクが起動される．この構成を図 9 に示す．

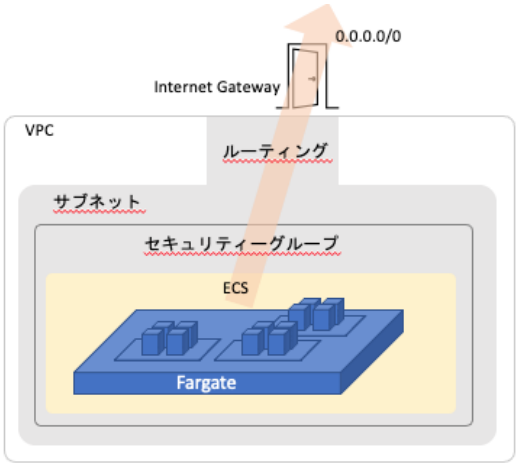

図 9 セキュリティー設計

5. 評価

2. 必要要件で定義した本システムが持つべき要件を実装されたシステムが満たしているかを評価する．

a) 認証機能

本システムの認証機能は JWT トークを使用することによって実現した． 既存システムとの一番の違いとしてはクラウドの分散環境， オートスケーリングを考慮に入れなければならない点である． このため， ステートレスな JWT トークンを Cookie に格納することによって認証機能を実装している．

b) MP4 ファイルのセグメントファイル化

ffmpeg による MP4 ファイルのセグメント化は Node.js によって構築されたサーバーサイドシステムによって実装される． ffmpeg は本来 CLI で操作されるが，fluent-ffmpeg の npm パッケージを使用することによって Node.js のプログラミングで処理の記述が可能となっている． そして， この npm には ffmpeg バイナリーファイルを登録することが可能でクラウド環境のシステムに自動で ffmpeg バイナリーがインストールされる．

c) 動画ファイルの管理

MP4 ファイルのセグメント化処理終了時に AWS が提供するオブジェクトストレージである Amazon S3 にセグメント化ファイルとそのメタファイルがアップロードされる． これによりクラウド環境で非推奨のディスクへの保存をせずに動画保存を実現している．

d) ビデオファイル検索機能

ブラウザでは Vue.js フレームワークによって作成された JavaScript が機能している． その機能の

一部としてユーザーが管理している動画内の検索処理を可能としている．

e) セグメントファイルのプレビュー機能

セグメント化された動画はその管理画面上でプレビューができるよう実装されている．セグメント化した全ての動画は動画一覧画面で確認でき，そこではセグメント化処理時に自動で生成されたサムネイル画像が使用される．このサムネイル画像をクリックすることでプレビューすることができる．

f) SCORM パッケージ化とそのダウンロード機能

e)と同じようにサムネイル画像をクリックするとその動画が SCORM パッケージ生成用動画として設定される．ダウンロードボタンが用意されており，それをクリックすると Zip ファイルがダウンロードされる．

g) クラウド環境での実装

ステートレスな JWT を認証に使用し，Amazon S3 を動画の保存場所としたことでクラウドの分散環境にも対応できる設計とした．

図 10 は完成したシステムのスクリーンショットである．

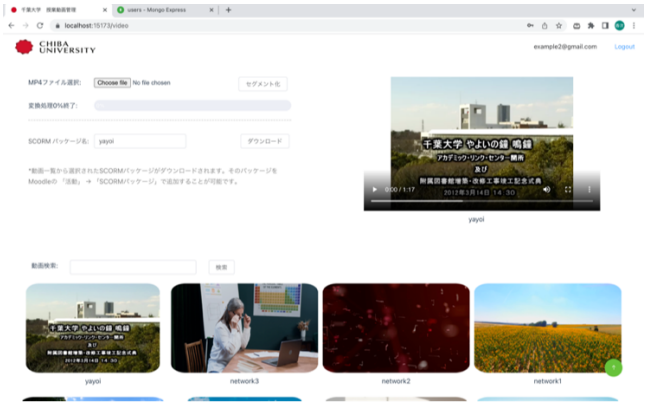

図 10 動画管理サイトのスクリーンショット

6. 結論

Moodle 用の動画管理システムの機能性を向上させるべく再設計を行った．そして，物理サーバーの管理を無くすことが可能なことから近年多くの機関に採用されているクラウド環境での実装を検討した．

1.2 目的 1 で定義したシステムの機能性に関してはサーバーサイド言語に Node.js を使用すること，Vue.js でクライアントサイドを SPA とすることで向上を図った．Node.js の導入は利用者人数が多い場合に複数のリクエストが送られることを想定したもので，この言語の強みである複数リクエストに対しての並行処理を利用し，効率的に処理されることが期待できる．Vue.js は SPA を利用することでユーザーの操作性の向上，サーバ負荷の軽減が期待できることから導入されている．このためサーバーサイドはページを生成せず API としてのみ使用される構成となった．

1.2 目的 2 で定義したクラウド環境での実装に際しては AWS を用いて行い，その実装を可能とするために Docker コンテナを用いた．分散環境であることから既存システムから設計を大きく変更することとなったがクライアントサイドに S3 の静的ホスティング，サーバーサイドに AWS Fargate を使用し，デプロイまで完了したことでこのシステムがクラウド環境で実装可能なことを示した．そして，クラウド上でのセキュリティー対策を行なうことでシステム使用の安全性を高め，ロードバランサーとオートスケーリングによる横方向スケーリングを可能としたことでトラフィックが増加した場合にも対応できるシステムとなっている．本システムの開発環境構築のために使用した Docker コンテナは AWS 以外でも実行することができ，GCS や Microsoft Azure などを採用したとしてもデプロイ可能である．

文　献

[1] 清水健一，藤本茂雄，松本暢平，檜垣泰彦，"Moodle 用動画 配信システムの内製とコロナ禍における運用・評価"，信学論 (D), Vol. J105–D, No. 10, pp. 572–583, July 2022.

[2] 原田明成，張徳鵬，檜垣泰彦，“メディア授業に適した授業動画 配信方式の検討,” 信学技報，vol. 119, no. 477, LOIS2019-67, pp. 63-67, 2020 年 3 月.

[3] HTTP Live Streaming, IETF, rfc8216, August 2017, https: //tools.ietf.org/html/rfc8216 （2023 年 1 月 20 日 閲覧）

[4] Ffmpeg, January 2023, https://ffmpeg.org/ffmpeg.html#Description （2023 年 1 月 20 日 閲覧）

[5] Amazon S3, January 2023, https://aws.amazon.com/jp/s3/ (2023 年 1 月 20 日 閲覧)

[6] Amazon Elastic Container Registry, January 2023, https://aws.amazon.com/ecr/ (2023 年 1 月 20 日 閲覧)

[7] Amazon Elastic Container Service, January 2023, https://aws.amazon.com/ecs/ (2023 年 1 月 20 日 閲覧)

[8] Amazon EC2, January 2023, https://aws.amazon.com/jp/ec2/ (2023 年 1 月 20 日 閲覧)

[9] Amazon Fargate, January 2023, https://aws.amazon.com/jp/fargate/ (2023 年 1 月 20 日 閲覧)