タイトル

　JGS研究2020プロジェクトチーム論文、IA001：

「サーバレスアーキテクチャーの適用によるシステム開発期間短縮効果の検証」について

ニッセイ情報テクノロジーの貴田とみずほ情報総研の松隈が発表致します。

アジェンダ

　本発表のアジェンダです。活動内容を簡単に説明したのち、問題提起、仮説の検証、課題、今後の展望の順でご説明いたします。

紹介

　活動内容です。本活動のテーマは「サーバレスアーキテクチャの適用検討」です。本テーマは昨年度から始まったテーマであり、昨年度の研究内容は「適用プロセスの検討」でした。

本研究では、サーバレスアーキテクチャの定義を昨年度の研究と同様のものとし、ビジネスアジリティ向上に対する是非を1年かけて検証いたしました。

問題提起①

問題提起です。IT業界のみならず様々な業種でデジタル技術を駆使したビジネスが創出され企業価値を高めている．こうした中で，各企業は競争力維持・強化のために，デジタルトランスフォーメーション（DX：Digital Transformation）をスピーディに進めていくことが求められている．

問題提起②

経済産業省のレポートによると、既存システムが老朽化・複雑化・ブラックボックス化する中では，メンテナンスの負荷が高く、ビジネスモデルの変化に迅速に対応できない、つまりビジネスアジリティを向上させることが困難である点が指摘されております。

近年、そのような状況を打破し、ビジネスアジリティを向上させる手段として、サーバレスアーキテクチャが注目されております。

サーバレスアーキテクチャの定義

サーバレスアーキテクチャの定義です。サーバレスアーキテクチャの定義はJGS2019研究プロジェクトチーム論文IA001：サーバレスアーキテクチャの適用検討プロセスの提案と同様としております。サーバレスとは「サーバ管理を必要としないアプリケーションの構築と実行の概念」の事を指します。表的な実現方法としてFaaS（Function as a Service）とBaaS（Backend as a Service）があります。

FaaSとは、ビジネスコードを実行する基盤そのものです。BaaSとは、多様なバックエンド機能をAPI ベースでサービスを提供するものです。

これらFaaSとBaaSの機能を組み合わせたシステム構成をサーバレスアーキテクチャーとしております。

サーバレスの特徴

サーバレスの特徴をサービス責任モデルを利用して説明します。

サーバやランタイム等のアプリケーションの実行に必要な基盤環境はクラウド事業者の責任の下、運用・管理されています。ユーザーは、アプリケーションコードとアプリケーションが扱うデータに注力することができます。

仮説

仮説です。本研究では、サーバレスアーキテクチャの適用により、ビジネスアジリティが向上することを仮説と致しました。ここで、ビジネスアジリティの向上とは、サービスリリースまでの開発期間が短縮されることをさします。本検証では、モデルシステムを実際にサーバレスアーキテクチャにて構築し、開発期間の短縮について、各開発工程のドキュメント作成量の減少、構築作業時間の2点から検証を行いました。

モデルシステム

構築したモデルシステムの題材です。本検証では、東京オリンピックのグッズ購買サイトをモデルシステムとして構築致しました。グッズ購買サイトとした理由は、政府がキャッシュレス決済を推進しており，ECサイトの立ち上げが増加すると考えたためです。実店舗を持つ企業が、東京オリンピックを契機として、販売チャネルの多角化による売上の拡大を行うためECサイトを立ち上げることを想定致しました。

モデルシステムに使用したリファレンスアーキテクチャ

モデルシステムに使用したリファレンスアーキテクチャです。検証作業に際し、グッズ購買サイトの要求仕様を纏め、その要求仕様にもとづき、AWS Samplesから類似のシステム構成を取得致しました。

モデルシステムのシステム構成

モデルシステムのシステム構成図です。

GraphQLサービスであるAppsyncを利用したWebシステムです。データベースにはDynamoDB、アプリケーションの実行にはLambdaを用いており、GraphQLのデータソースにはDynamoDBのほかに、HTTPエンドポイントとしてAPIgateway、StepFunctionsを使用しています。

グッズ購買システム：カタログ参照機能

今回の構築作業では、商品カタログ参照、ポイント参照、商品購入の３つの機能を実装しました。各機能について説明していきます。

商品カタログ参照機能です。利用者が選択した商品IDをキーに商品情報を収集します。

GraphQLには複数のデータベースやマイクロサービス、そして API に対し、単一の GraphQL エンドポイントから迅速にクエリができる特徴があります。本機能では、GraphQLのデータソースに、DyanamoDBを１テーブルを指定しています。クライアントからAPIリクエストを受信すると、バックエンドのDB１テーブルに対してクエリを行う、シンプルな構成です。

リクエストはHTTP GETメソッド、データ交換フォーマットはテキストベースのJSONを利用しています。

（続いて）グッズ購買システム：ポイント参照機能です。

利用者のユーザーIDをキーに保有ポイント数を取得する機能になります。

〜書きかけ〜

「構築作業時間」の計測の手順

仮説の検証作業について説明します。

・構築作業時間の計測

・ドキュメント作成量の計測　の順番で説明します。

「構築作業時間」の計測の手順です。まず、事前学習として、AWS開発者ガイドやチュートリアルを参照し、AWSサービスについて学習しました。次に要求仕様に基づき、データ定義書と各AWSサービスのパラメーターシートを作成した後、グッズ購買サイトのアプリケーション開発、基盤の構築を行い、構築時間を計測しました。

なお、アプリケーション開発と基盤の構築はAWSを利用した開発の未経験者で実施しました。

結果

計測結果です。コーディングは7時間、インフラの構築は3時間でした。

「ドキュメント作成量」の計測の手順

「ドキュメント作成量」の計測の手順です。

メンバーが開発経験のあるウォーターフォールモデルによる開発に必要な成果物を基準とします。

まず、オンプレミス環境でウォーターフォールモデルによる開発に必要となる成果物を定義しました。その後、その定義に基づき、サーバレスアーキテクチャーによる開発で必要となる成果物を別に設定し、オンプレミスとサーバレスアーキテクチャ適用におけるドキュメント作成量を比較、ドキュメント作成量が減少した項目について、根拠の整理を行いました。

結果

計測結果です。ドキュメント作成に掛かる作業量が減少したものが26%、ドキュメント作成が不要となるものが18%でした。

考察

考察です。

今回、構築作業時間とドキュメント作成量が減少した要因は2つあると考えています。

1つ目は各Lambda関数が「目的が1つもしくは2つで，ステートレス」な設計となっていることです．関数で実現することが簡素なので，プログラムステップ数が少ない．そのため，コーディング時間が短くなり，かつテストもし易くなります。更に，ステートレスであることから、セッション情報の管理を考慮する必要も無くなるため、開発時間が削減されたと考えます。

２つ目は非機能要求部分の実装自動化によってユーザーの設計・開発の範囲が限定的となることです。サーバレスでは最初に述べた通り、サーバやランタイム等のアプリケーションの実行に必要な基盤環境は，クラウド事業者の責任の下，運用・管理がされているため、ユーザの設計・開発の範囲が限定的になります。ドキュメント作成量、構築作業量が削減されたと考えます。

業務ロジック及びアプリケーション開発に対しては，開発期間の短縮効果は確認できなかった．サーバレスとは「サーバ管理を必要としないアプリケーションの構築と実行の概念」である．対象業務の現状を分析し，新たに実現すべき業務の流れを明確化するための業務要件定義や，アプリケーションの構造・設計，利用するデータの設計はユーザーの責任範囲である．

結論

結論です。検証結果から、サーバレス適用によりビジネスアジリティが向上すると結論づけました。

課題

課題です。課題には、複数言語の保守と人材育成が挙げられます。まず複数言語の保守についてです。Lambdaでは、複数言語をサポートしています。複数言語を選択できることはメリットでもありますが、複数言語の保守が必要となります。

課題

人材育成です。

検証では、システム構築にあたり事前学習を述べ1か月行った。学習内容はコーディング、環境構築、仕様理解の3点です。

エンタープライズレベルのシステムを構築するのであれば、さらに膨大な知識とスキルが必要になるため、相応の教育体制が必須となります。

また、AWSでは2020年3月においても175を超える主要なサービスがあり、日々新しいサービスが開発されている。そのため、継続的な学習も要求されます。

さらに、有識者が存在していたとしても，各部門・領域に点在している状態では，組織としてエンタープライズレベルの開発を推進していくことは困難です。そのため，クラウドを用いた開発を進めていくためには，ノウハウを集約した専門組織を設立し，領域横断的な人材育成を行うことが望ましいです。

今後の展望

今後の展望です。今回、検証はしておりませんが、サーバレスアーキテクチャを適用することで、仮想化以下のインフラ管理と非機能要求に付随する作業量が減少し、結果、運用業務が省力化されると考えます。そのため、運用に割いていたコスト・人員を開発業務に充てることができ，よりスピーディな開発が可能になると考えます。

今後の展望

また、開発時に。各アプリケーションを疎に結合させることで，つまりマイクロサービス化させることで、機能追加・仕様変更時の際も対応範囲を局所化でき，更に低コスト・短納期で開発が可能であると考えます。また上記の観点に加え、アジャイル開発を組み合わせることで，より大きな効果が期待されると考えます。