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、今後の展望の順で説明します。

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年かけて検証しました。

背景

昨今、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ます。こうした中で，各企業は競争力維持・強化のために， DXをスピーディに進めていくことが求められていますが、計画的に取り組まれているでしょうか？

問題提起

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います。ちなみにスライドにはポンペイ遺跡が移っていますが、この遺跡にように老朽化・複雑化し、少しの修繕でも影響範囲が未知数なレガシーシステムがDXの推進を阻害しているケースも多いのではないでしょうか。

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います！！

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としております。

サーバレスの特徴

AWSオンラインセミナーなどでご覧になった方もいらっしゃるかと思いますが、サーバレスの特徴について、サービス責任モデルを利用して説明します。

ご経験されている方も多いと思いますが、オンプレでシステムを作成する際は、ネットワークやサーバなどの物理的なものから、アプリには欠かせないミドルウェア、アプリの根幹となるアプリケーションコードやデータなどのすべてをユーザーの責任で構築・運用する必要があります。

サーバレスでは、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・管理されているため、ユーザーは、これまで基盤環境に割いていた時間を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

一般的に、サーバレスアーキテクチャを適用することで、ビジネスアジリティが向上するといわれています。本研究では、それを仮説とし、サーバレスアーキテクチャを適用することによって、実際にビジネスアジリティが向上するのか検証しました。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、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

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の開発を容易にする完全マネージド型サービスである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を用いています。Appsyncの呼び出し先となるデータソースには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。

（言葉にしない→）（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。

AppSyncの内部で動いているGraphQLというサービスには、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を実行できる特徴があります。この機能は、Appsyncのデータソースに、DyanamoDBの１テーブルを指定しており、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DB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、AppSyncとDynamoDB間のデータ受け渡しは、JSON形式のデータを利用しています。

（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。

AppSyncのデータソースには、API Gatewayエンドポイントを指定しています。

AppsyncはクライアントからAPIリクエストを受信すると、API Gatewayで作成したREST APIを利用してデータ取得を行います。業務ロジック部分は、Lambdaを実行基盤としてpythonでコーディングされたもので、DynamoDBからデータ取得を行います。

（最後に）グッズ購買システム：商品購入です。

利用者が選択した商品の購入・決済処理を行います。本機能は、6ステップのアプリケーション・ワークフローで構成されています。ワークフローの実行には、LambdaやAWSの複数のサービスを自動で実行・管理するAWS Step functionsを利用します。

AppSyncのデータソースに、Step functionsのエンドポイントを指定しており、アプリはStep functionsと紐付いている複数のLambda関数が行います。

クライアントからAPIリクエストを受信すると、Step functionsのステートマシンを起動して非同期処理でアプリケーション・ワークフローを実行します。Step functionsの処理結果は、同処理結果をサブスクライブしているクライアントへクエリの結果が通知される仕組みです。　　・・・この説明難易度高い

「構築作業時間」の計測の手順

検証作業では、構築作業時間の計測、ドキュメント作成量の計測をそれぞれ実施しました。まずは構築作業時間の計測から説明します。

検証にあたり、このような要求仕様書を作成しました。(スライドを足す)

「構築作業時間」の計測にあたって、まず事前学習を行いました。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

その結果、なんとコーディングは7時間、インフラの構築は3時間でした。明確に削減効果が現れました。

「ドキュメント作成量」の計測の手順

つづいて「ドキュメント作成量」の計測です。

ここでは、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した。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。

要求仕様書

パラメーターシート

結果

その結果、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100%としたとき、サーバレスアーキテクチャを適用した開発においては、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。ドキュメント作成量においても明確に削減効果が表れています。

考察

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くなります。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。ちなみに今回の検証においても、構築したLambdaは7本であり、ステップ数はおよそxxでした。そのため、7時間という短期間での開発が可能になったと考えます。

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、ユーザーの設計・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ませんでした。サーバレスとは先ほど説明した通り、「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」です。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となるため、短縮効果は得られないと考えます。

結論

このような検証結果から、我々は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

ただし、いいことばかりではなく、課題もあります。複数言語の保守と人材育成の2点です。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

続いて、人材育成についてです。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。

総括

ということで、サーバレスアーキテクチャを適用することで、ビジネスアジリティが向上するのかという点について、我々はサービスリリースまでの開発期間が短縮されることで、迅速なサービス提供が可能となると結論づけました！以上です！

活動について

（一呼吸あけて、テンションを変え、さっぱりした感じで）ここからは、活動の振り返りと所感です。

今年度の活動ですが、新型コロナ感染症の影響があり、集合形式の活動が実施できなくなり、論文執筆を大きく進める合宿も開催できませんでした。3月以降は、WEB会議主体に活動を進めましたので紹介したいと思います。

無料で利用可能なツールを3つ利用しました。

コミュニケーションツールはslackを利用しました。Slackについては、新型コロナ以前も利用していました。

ドキュメント、資料の共有にはGithubを利用しました。パブリックのリポジトリを作成し、論文ファイル、モデルシステム構築時のソースコードを共有し、作業を進めました。

ビデオ会議ツールは「Jitsti」を利用しました。時間制限がなく、画面共有機能、チャット機能もありました。

最後に所感です。