https://github.com/MikioKaya/JGS\_Serverless/tree/master/英語論文

形式：起承転結

タイトルスライド

1. 紹介
   1. 研究内容・結論 or 疑問を提示するまで

サーバーレスについては、良いことばかり言われているが、

サーバレスアーキテクチャーを適用することでビジネスアジリティが向上するのは本当か？

1. 問題意識の提起
   1. DX事例紹介

DXに取り組むことでビジネスの競争力維持・強化につなげる。

事例を取り上げる。

先行する企業に対抗するために、何に取り組んでいけばよいかの

* 1. 現状紹介

DXに取り組む前の現在（レガシー）について説明

* 1. 対策の紹介

レガシーシステムを切り崩す新たなデジタル技術としてサーバレスが注目されている。

1. サーバレス、サーバレスアーキテクチャー定義
   1. 用語解説

サーバーレス　の定義

サーバレスアーキテクチャー　の定義

* 1. ユーザ／クラウド事業者の責任モデル

サーバーレスの責任モデル

　オンプレ、IaaS、PaaS、サーバレス

1. 検証
   1. 仮説

開発期間が短縮、観点は以下の通り

　 ・構築作業時間

　 ・ドキュメント量

※オンプレ開発については、これまでのシステム開発の知見に基づき設定。

* 1. 検証方法
     1. 検証方法

ビジネスアジリティの向上をどのように検証するのか？

今回の検証は、AWSを利用して、グッズ購買サイトをサーバレスアーキテクチャーで構築する。

⇒理由も簡単に(今後売り上げを増やしていくために、新規にECサイトの基盤を立ち上げる状況を想定)

（グッズ購買システムの設定理由は、口頭で説明予定）

* + 1. グッズ購買サイトのシステム構成図

（AWS構成図添付）

* 1. 構築作業時間
     1. 構築作業

検証方法は、以下の通りとする。

【構築作業時間】

手順Aー1→Aー2→・・・（フロー図にまとめる想定）

* + 1. 結果

（表で各種時間をまとめる）

・事前学習（A-1、A-2）　延べ1ヵ月

・作業結果（A-3、A-4）　１０時間、コーディング７時間、環境構築３時間

* + 1. 考察

(1)環境構築　３時間

　サーバレスな各サービスを選択・連携してシステムを実現するため、ユーザーの 　設計範囲が限定的

　→特に非機能要件部分について、クラウド事業者が責任範囲が大きく、開発範囲　も限定されるため、開発作業時間が短縮された。

（サービス責任モデルを添付しても良いかも）

(2)コーディング　7時間

　Lambda関数で実現することが簡素かつステートレス

　→各Lambda関数は「目的が1つもしくは2つで，ステートレス」な設計 となる。　ステップ数も少なく、セッション管理も行わないため、テストが容易。

(3)事前学習の苦労話も伝えたい

* 1. ドキュメント作成量
     1. 作業

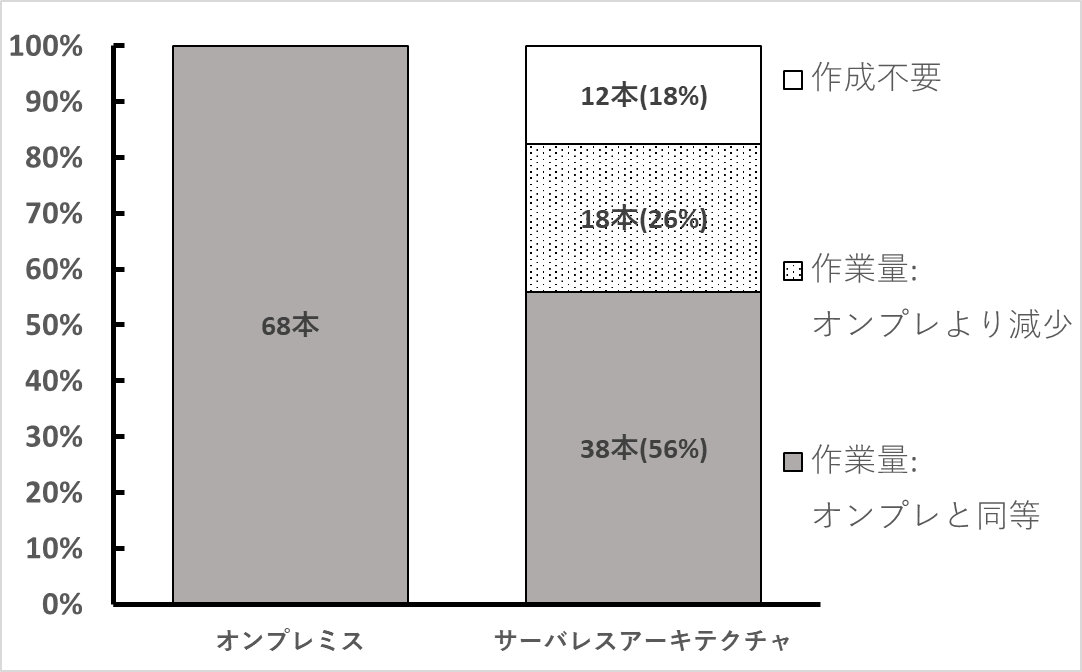
【ドキュメント作成量】

手順Bー1→Bー2→・・・（フロー図にまとめる想定）

成果物一覧のイメージを添付（一部分を切り取って作成する）

* + 1. 結果

（論文３章の表を引用）



* + 1. 考察

(1)サーバレスな各サービスを選択・連携してシステムを実現するため、ユーザーの　設計範囲が限定的

　評価シートのどの工程・作業が主に作成量が減ったのか説明する。

　→特に非機能要件部分について、クラウド事業者が責任範囲が大きく、ユーザー　　の設計範囲が小さいため、ドキュメント量も減少

(2)サーバレスアーキテクチャーの適用により作業量が減少しない作業

　 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．

　→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り、従来のオンプレ同様に必要な作業となる。

1. 検証結果まとめ
   1. 検証作業結果のサマリ

４章の各作業結果を一スライドに纏める

* 1. まとめ

サーバレスアーキテクチャーを適用することで，サービスリリースまでの開発期間が短縮され，迅速にサービスの提供が可能になる．

結果，ビジネスアジリティが向上すると言える．

1. 課題
   1. 開発言語の選択

適切な言語を選択する自由度が増す反面，保守性が低くなるため，開発工程前に言語を選定するのが望ましい

Lambdaがサポートするランタイム

OS：Amazon Linux　or　Amazon Linux2

ランタイム：Node.js / Python / Ruby / Java / Go / .Net

カスタムランタイム：

　シェルスクリプト、Amazon Linux に含まれる言語のスクリプト、または Amazon Linux 上でコンパイルされたバイナリ実行ファイルを実行することができます。

・実績ベース

　Python / Node.jsを利用して実装した

* 1. クラウド人材の育成

構築作業前の事前学習に延べ1ヶ月程度費やした。何を学習したか。  
 ・AWSの基本概念，各サービス仕様等

・AWS 初めてのサーバレスウェブアプリケーション構築

<https://aws.amazon.com/jp/serverless/build-a-web-app/>

・AWSマネージドコンソール共有アカウントを作成してチュートリアル実施

Appsync、API Gateway 、Stepfunctions 、Lambda等

　⇢着手が遅れた・・もったいない

①AWSが提供するサービスは日進月歩で進化している。

　現場のエンジニアは当然，意思決定権のある経営者においても知識の習得が必要

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難しい．

②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．

1. 今後の展望

今回検証できていない以下の2つの観点を追求することで，更にビジネスアジリティを向上

させる可能性がある観点は以下です。

* 1. 開発業務への注力

サーバレスアーキテクチャーを適用し運用業務を省力化することで，運用に割いていたコスト・人員を開発業務に充てられるため，よりスピーディな開発が可能になると考える．

⇢ 従来のハード・ソフトの保守を担当していた人員は役割が変化する。

　従来の役割の比率が徐々に下がり、新たな役割が増えていくでしょう。

　組織レベルでは業務内容の再構築が必要となり、

　（当然だけど）個人は　新たに技術を学んでいく必要あり。中堅以上の技術者が、クラウ

　ドネイティブな考えを受け入れていく必要がある。既存技術の代替として捉えると誤った

　結論を得てしまうよ。

* 1. アプリケーションのマイクロサービス化

各アプリケーションを疎に結合させることで，機能追加・仕様変更時の際も対応範囲を局所化でき，低コスト・短納期で開発が可能であると考える．また上記の観点に加え，ドメイン駆動設計や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．

　　　7.3 まとめ

繰り返しだけど、

開発と運用の一体的体制やアジャイル開発を推進する組織風土の醸成に加え，

高度な技術力とクラウドに対する豊富な知識を持つ人材の育成が必要である．

説明はここまで。

1. 所感

スライド作ってもよいかも

1. メンバー紹介

集合写真のスライド