ITR White Paper

「2025年の崖」から落ちないための アプリケーション変革

～マイクロサービス、DevOps、コンテナを採用したクラウドネイティブへの道～

株式会社アイ・ティ・アール

C19110126

目 次

第1章　エグゼクティブ・サマリ

経済産業省が2018年9月に発行した『DXレポート ～ITシステム「2025年の崖」克服とDXの本格的な展開～』（以下、DXレポート）が国内企業に与えた影響は大きい。このレポートの趣旨は、以下のようなものである。全ての産業において、先進的デジタルテクノロジを活用した革新的なビジネスモデルを展開する企業が参入してゲームチェンジが起きているが、競争力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全ての企業に求められている。企業はDXの重要性は理解し、DX推進のための組織を作るなどしているものの、その多くはビジネス成果につながっていない。その大きな要因のひとつとして「レガシーシステム」（経済産業省では、技術面の老朽化、システムの肥大化・複雑化、ブラックボックス化などの問題があることによって、経営・事業戦略上の足かせ、高コスト構造の原因となっているシステムのこと、と定義している）の存在をあげており、これにより、データを十分に活用できずDXを実現できないため、デジタル競争の敗者になると述べている。敗者にならないためには、2020年までにシステム刷新の経営判断を行い、2021年から2025年にかけてビジネス戦略を踏まえたシステム刷新を経営最優先課題としてシステム刷新を断行するように提言している。このような経営判断ができず、システム刷新を計画的に実施できない企業は「2025年の崖」から転落し、デジタル競争に敗れ消え去っていく可能性が高い。

「2025年の崖」から落ちないためには、戦略的に自社の既存アプリケーションをDXに活用できるものに再構築したり、ユニークなアイデアを実現するためのアプリケーションを迅速に新規構築する、といった変革が必要である。これに対応するにはアプリケーションをクラウドネイティブにすることが最も有効である。多くの国内企業は、コスト削減のために自社所有サーバやデータセンターを代替するサービスとしてクラウドを利用している。しかしクラウドは、アプリケーションを迅速に開発し、利用状況に応じて自動的にリソースの伸縮が可能で、低コストで無停止運用や高可用性を実現できる革新的テクノロジである。既存アプリケーションをクラウドに移行するだけでそのようなメリットを享受することは困難であるため、企業は、DXに貢献する、つまり、イノベーションや差別化に寄与するアプリケーションには、クラウドネイティブ・アーキテクチャを採用すべきである。

全ての企業アプリケーションにクラウドネイティブ・アーキテクチャを採用することは非現実的であるため、企業は自社アプリケーションの特性を評価し、どのように

クラウドを利用するのがよいか事前にアセスメントを行い、クラウドネイティブ・アプリケーションを採用する業務を検討すべきである。既存アプリケーションをそのままクラウドに移行する場合でも、ITインフラを標準化することができ、統合運用や自動化が可能になり、グローバルでの展開も容易になるため、大きなメリットを獲得することができる。また、アプリケーションのクラウドネイティブ化では、マイクロサービス・アーキテクチャ、DevOps、Kubernetes、サーバレスといった重要テクノロジを理解したうえで適切に活用することが必要である。

第2章　クラウドネイティブのビジネス価値

クラウドは、アプリケーションを迅速に開発し、利用状況に応じて自動的にリソースの伸縮が可能で、低コストで無停止運用や高可用性を実現できる革新的テクノロジである。しかし、既存アプリケーションをクラウドに移行するだけでは、そのようなメリットを享受することは難しい。企業は、イノベーションや差別化を行うためのアプリケーションについては、クラウドネイティブ・アーキテクチャを採用すべきである。

国内企業アプリケーションの現状

ますます激しくなるグローバル競争、国内市場の飽和感や閉塞感、顧客の価値観の変化と多様化など、国内企業を取り巻く状況は厳しく、何らかのブレイクスルーを探索する企業が増えている。こうした背景から、近年、DX（デジタルトランスフォーメーション）に取り組む国内企業が増えている。

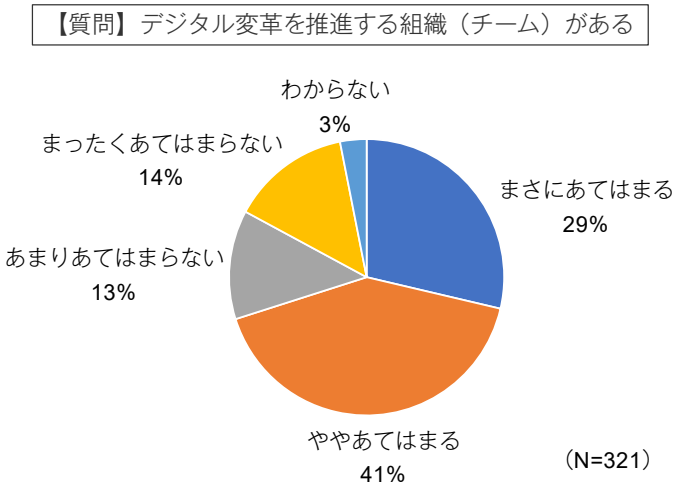
DXの概念は、2004年にスウェーデンのウメオ大学のエリック・ストルターマン教授が提唱したとされている。ストルターマン教授は、「ITの浸透が、人々の生活をあらゆる面でより良い方向に変化させる」ことがDXである、と述べている。

この抽象的な定義では、自らの行動に反映することが難しい。DXに対する取り組みをより的確に表しているものとしては、経済産業省が2018年12月に発表した「DX推進ガイドライン」が参考になる。このガイドラインでは、DXとは「企業がビジネス環境の激しい変化に対応し、データとデジタル技術を活用して、顧客や社会のニーズを基に、製品やサービス、ビジネスモデルを変革するとともに、業務そのものや、組織、プロセス、企業文化・風土を変革し、競争上の優位性を確立すること」と定義している。

図1に、ITRが実施した国内企業におけるDX動向調査の結果を示す。IT部門内、事業部門内、専門組織の別を問わず「デジタル変革を推進する組織（チーム）がある」と回答した企業は7割に上り、DXに前向きな企業が多いことがわかった。

しかし、現実には、ほとんどの企業が老朽化したアプリケーションを抱えており、それらがDXの足かせになっている。

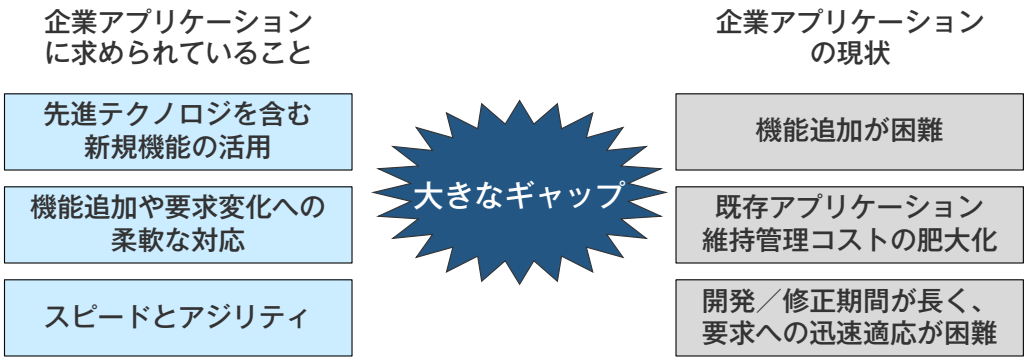
図1　デジタル変革を推進する組織（チーム）の設置状況



出典：ITR「デジタルトランスフォーメーション成熟度調査」（2019年6月）

既存の老朽化したアプリケーションがネックになっているのは、DXのようなイノベーション領域だけではない。定常業務においても、顧客要求や外部環境の変化はますます激しくなっており、アプリケーション開発におけるスピードやアジリティに対するユーザーからの要求は高い。しかし、多くの企業の既存アプリケーションはこのような要求に応えられていないといっても過言ではない。現在企業アプリケーションに求められていることと、現状との大きなギャップを図2に示した。

図2　企業アプリケーションに求められていることと、現状との大きなギャップ



出典：ITR

このようなギャップが発生したのは、アプリケーションのアーキテクチャに起因するところが大きい。現代のアプリケーションに求められている「先進テクノロジを含む新規機能の活用」「機能追加や要求変化への柔軟な対応」「スピードとアジリティ」

を実現するための有力なアプローチのひとつが、「クラウドネイティブ・アプリケーション」である。

「クラウドネイティブ・アプリケーション」とは何か

クラウドネイティブ・アプリケーションの特徴を図3に示した。図には、従来型、つまり非クラウドネイティブ・アプリケーションの課題も併記した。従来のアプリケーションはアプリケーション全体が巨大構造になっており、先進テクノロジの組み込みやプログラムの修正が容易ではなかった。しかし、クラウドネイティブ・アプリケーションは「マイクロサービス・アーキテクチャ」になっており、機能追加や修正が容易に行える。また、クラウドネイティブ・アプリケーションは、迅速なインテグレーション／デプロイやプロビジョニングが可能で、最小の人的作業で、または自動的にスケーラビリティや冗長化が実現できる。このような構造のアプリケーションが実現できれば、現在のアプリケーションが抱える課題のほとんどが解消できる。

図3　クラウドネイティブ・アプリケーションの特徴

特徴	クラウドネイティブ・アプリケーション	非クラウドネイティブ・アプリケーション
マイクロサービス・アーキテクチャ	アプリケーションが小さな粒度のサービスの疎連携で構成されており、いつでも別のサービスに入れ替えることができる。先進テクノロジのアプリケーションへの組み込みやプログラム修正が容易である	アプリケーション全体が巨大なプログラムで構成されており、先進テクノロジの組み込みやプログラムの修正が容易でない
迅速インテグレーション／デプロイ	DevOpsやCI／CDツールと組み合わせて、「コーディング→ビルド→テスト→デプロイ→運用」が極めて迅速かつ高頻度に実行できる	DevOpsやCI／CDツールなどを活用できるが、テストやデプロイに時間がかかり、無停止でのデプロイも困難であるため、高頻度でシステムを更新することができない
迅速プロビジョニング	ユーザーが必要とするタイミングでアプリケーションを立ち上げることができる。また、その際にアプリケーションに必要なリソースを迅速に、時には自動的に割り当てることができる	アプリケーション・デプロイ作業に時間がかかるため、その開始時期は事前にユーザーと調整する必要がある。アプリケーションに必要なサーバ、ストレージ、ネットワーク、ミドルウェアなどは事前に入念に設計し、個々にインストールおよび設定を行わなければならない
自動スケーラビリティ	事前サイジングは不要で、最小限のリソースでアプリケーションを開始できる（スモールスタート）。アプリケーションの利用状況に応じて自動でリソースを伸縮させることができる	事前の入念なサイジングが必要だが、確保したリソースを超えるアクセスが発生した場合、急激なパフォーマンス低下を招く。割り当てリソースの拡大／縮小にはアプリケーションを停止する必要がある
自動冗長化	アプリケーションそのものに冗長化機構が組み込まれており、リソースの一部に障害が発生しても、アプリケーションが停止しない	アプリケーションそのものには冗長化機構はないため、ITインフラに冗長化の工夫を施す。自動回復は困難

出典：ITR

クラウドネイティブ実現のためのキーワード

前述の通り、クラウドネイティブ・アプリケーションは非常に優れたアーキテクチャであるが、その実現は容易ではない。次章以降に、クラウドネイティブ・アプリケーションの実現に向けたアプローチや留意点を述べるが、ここでは、それらに対する理解を促進するために重要なキーワードを簡単に説明しておく。

コンテナ

サーバ仮想化には「ハイパーバイザ型仮想化」と「コンテナ型仮想化」がある。ハイパーバイザ型仮想化とは、仮想マシンを個別に立ち上げ、その中でゲストOSとしてサーバOSが稼働するものである。コンテナ型仮想化とは、ホストOSから見ると1つのプロセスと見なされる「コンテナ」を利用する。コンテナにはゲストOSは含まれず、ライブラリもアプリケーションを稼働させるための最小構成ソフトウェアを梱包することが多いため、ITリソース（CPU、メモリ、ストレージ）の消費量が非常に小さくて済む。オーバーヘッドが極めて小さいため、コンテナの起動は非常に速い。コンテナツールは数多く存在し、代表的なものがDockerである。

コンテナ・オーケストレーター

コンテナを統合運用管理するソフトウェアを総称して「コンテナ・オーケストレーター」と呼ぶ。代表的なものはKubernetesである。Kubernetesは、Google社がGmail、YouTubeといった大規模サービスを効率よく運用するために独自開発し、10年以上運用に活用してきたツールをOSS化したものである。Kubernetesは、オンプレミス・サーバやIaaS上にインストールし、自社でその運用保守を行う形態も可能であるが、Kubernetesのフルマネージド・サービスがAmazon Web Services社、Google社、Microsoft社など多くのクラウド事業者からも提供されている。

マイクロサービス

「マイクロサービス」とは、適度な粒度で構成されるサービス群とサービス間連携にAPIを利用するアーキテクチャを指す。個々のサービスは他サービスの内部プログラムやビジネスロジックを知る必要がなく、必要な情報をAPI経由で問い合わせて情報を入手すればよい。また、個々のサービスは他のサービスから完全に独立しており、APIの仕様を変えない限り、サービス内のプログラム変更やロジック変更が他のサービスに影響しない。手組み／パッケージ／SaaSなどのアプリケーション開発形態、開

発言語／フレームワーク／高速開発ツールなどの実装方法がサービスごとに異なっても、アプリケーション全体の動作には問題がない。

サーバレス

「サーバレス」とはユーザーがサーバの存在を気にすることなく、アプリケーション開発に集中できるサービスを指す。サーバレス・アーキテクチャを実現するのは、データベース／マルチプラットフォーム対応開発環境／認証など、企業アプリケーションに必要な環境を統合提供するBaaS（Backed as a Service）と、イベントによりトリガーされステートレスなコンテナ上で実行される短命（概ね1回の呼び出しごと）なフルマネージド・サービスであるFaaS（Function as a Service）がある。現在、より注目されているのは後者である。

DevOps

アプリケーション構築、機能追加／修正、運用において、変化に迅速に適応でき、品質を向上させ、コストを削減するための手法を、DevOpsと呼ぶ。DevOpsを実現するには、全体を考えた自律的な思考と活動をする企業／組織の文化（Culture）を作ること、無駄を極小化するリーン（Lean）な活動を行うこと、自動化（Automation）を積極的に推進すること、常にアプリケーションの状況を定量化し常時測定（Measurement）すること、関係者全体の情報共有（Share）を積極的に推進することが求められる。

第3章　クラウド・アセスメント

自社アプリケーションの全てにクラウドネイティブ・アーキテクチャを採用することは理想的だが、数多くのアプリケーションを抱える企業にとってそれは現実的ではない。既存アプリケーションのクラウド移行、および新規アプリケーションのクラウド上での開発にはパターンがあり、アプリケーションの特性によって使い分けるべきである。企業は、事前に対象アプリケーションのクラウド・アセスメントを行って最適なアーキテクチャを採用すべきである。

クラウド活用パターン

クラウドには、事業者がITリソースを所有しサービスとしてITインフラや開発環境を提供する「パブリッククラウド」と、ユーザー企業がITリソースを自社のリスクで保有（買取り、リース、レンタルなど）してITインフラを利用する「プライベートクラウド」が存在する。本稿では、「パブリッククラウド」を「クラウド」と、「プライベートクラウド」を「オンプレミス」と記述する。なお、クラウド・アーキテクチャを採らない単体のサーバ・ハードウェアは、今後企業にとって極めて価値が小さいとITRでは考えていることから、本稿では議論の対象外とする。

企業がクラウドを活用するパターンとしては、既存アプリケーションをクラウドに移行するケースと、クラウド上で新規アプリケーションを構築するケースの2通りがあるが、どのようなアプリケーションでもクラウドを利用すべきというわけではない。図4にクラウドとオンプレミスの判断のためのガイドラインを示す。

図4　クラウド／オンプレミスの判断ガイドライン

クラウド（パブリッククラウド）	オンプレミス（プライベートクラウド）
• リソースの柔軟性や拡張性を必要とするアプリケーション • アクセス数や負荷が大きく変動するアプリケーション • 運用前にアクセス数や負荷を予測することが困難なアプリケーション • 主にインターネットから利用するアプリケーション • グローバル規模で利用するアプリケーション • 短期間しか利用しないアプリケーション • 高度なセキュリティレベル（PCI-DSS、FISCなど）を必要とするアプリケーション	• ハードウェアとの密接な連携が必要なアプリケーション • ミリ秒単位のレスポンスが必要なアプリケーション • 既存オンプレミス・システムとの密接または多数の連携が必要なアプリケーション • エッジコンピューティング用アプリケーション • クラウド上での稼働が保証されていないパッケージやミドルウェアを利用しているアプリケーション • 法規制でクラウド利用が禁止されているアプリケーション • 政府機関またはそれに準じる組織において機密データを保管しているアプリケーション

出典：ITR

製造機器、検査機器や物流機器といったハードウェアは、単体で稼働していることは少なく、アプリケーションで制御／管理／監視を行っていることが多い。このような機器は、特殊なネットワークや信号伝送ケーブルを介してアプリケーションと接続されているため、クラウドでアプリケーションを稼働させることは非現実的である。メインフレームやオフコンなどのオンプレミス環境で稼働しているアプリケーションに対して、極めて短い時間（ミリ秒など）のレスポンスでデータを送受信する必要があるアプリケーションも、クラウドに適さない。IoTのエッジコンピューティングには、遅延が極めて小さいことが求められるため、クラウド上への設置は容易ではない。また、政府機関では、各府省庁情報化統括責任者（CIO）連絡会議が2018年6月に発行した「政府情報システムにおけるクラウドサービスの利用に係る基本方針」の「クラウド・バイ・デフォルト原則」において、『特定秘密および行政文書の管理に関するガイドラインに掲げる秘密文書中極秘文書に該当する情報はパブリッククラウド上で扱わないものとする』と明記している。つまり、政府機関やそれに準じる組織においては、特定秘密／極秘文書を取り扱うアプリケーションはクラウドで運用できないと考えたほうがよい。

クラウド移行パターン

既存アプリケーションをクラウドに移行するパターンを図5に示した。最もシンプルなパターンは、Rehost（リホスト）である。これは、既存アプリケーションのアーキテクチャを変更せずに、サーバを現行環境（仮想サーバまたは単体サーバ）からIaaSに移行することを指す。現行環境のうち、ハードウェアに依存している部分（ファイバーチャンネル接続やサーバ・ハードウェア接続による冗長化など）を修正するだけでよいため、国内の企業ではよく利用されているパターンである。

Refactor（リファクタ）は、既存アプリケーションのアーキテクチャは変更せずに、データベース・ソフトウェアをマネージドサービスに変更したり、アプリケーション全体をコンテナ化するなど、部分的なインフラ変更によってクラウドの利点をより活かせる構造にすることを指す。

Rearchitect（リアーキテクト）とは、既存アプリケーションを主要機能ごとに分割し、コンテナ化やマイクロサービス化といった手法でアーキテクチャを変更し、クラウドネイティブ・アプリケーションを実現することを指す。

Rebuild（リビルド）とは、既存アプリケーションのロジック（プログラム）およびアーキテクチャをゼロベースで見直し、クラウドネイティブ・アプリケーションとして再構築することを指す。

Replace（リプレース）とは、既存アプリケーションを破棄し、同等の機能を提供するSaaSに移行することを指す。

図5　既存アプリケーションのクラウド移行パターン

移行パターン	概要	利用機能／サービスの例
Rehost （リホスト）	既存アプリケーションのアーキテクチャは変更せず、サーバを現行環境（仮想サーバまたは単体サーバ）からIaaSに移行すること	仮想サーバ・サービス（IaaS）
Refactor （リファクタ）	既存アプリケーションのアーキテクチャは変更せず、部分的にインフラ・モジュールを変更したり（マネージド・データベース利用など）、インフラ構成を変更したり（コンテナ利用など）して、クラウドの利点をより活かせる構造にすること	コンテナ・オーケストレーター 仮想サーバ・サービス（IaaS）
Rearchitect （リアーキテクト）	既存アプリケーションのアーキテクチャを変更し、クラウドネイティブ・アプリケーションを実現すること	マイクロサービス コンテナ・オーケストレーター
Rebuild （リビルド）	既存アプリケーションのロジックおよびアーキテクチャをゼロベースで見直し、クラウドネイティブ・アプリケーションに再構築すること	サーバレス マイクロサービス コンテナ・オーケストレーター
Replace （リプレース）	既存アプリケーションを破棄し、SaaSに移行すること	SaaS

出典：ITR

続いて、図6にそれぞれのクラウド移行パターンのネイティブ度を比較した。Rehostは、クラウドネイティブ要件をまったく満たさない。Refactorでは、「迅速インテグレーション／デプロイ」および「自動冗長化」は実現可能であるが、その他の要件を満たせずクラウドネイティブとはいえない。Rearchitectは、工夫次第で「迅速プロビジョニング」は実現可能であり、その他の要件を満たすことができるため、クラウドネイティブ度は高いといえる。Rebuildでは、完全なクラウドネイティブ・アプリケーションが実現可能である。もちろん、どのような実装内容でもクラウドネイティブ・アプリケーションができるわけではなく、そのための事前の入念な設計と実装スキルが必要となる。

図6　クラウド移行パターンのネイティブ度比較

移行パターン	マイクロサービスアーキテクチャ	迅速インテグレーション／デプロイ	迅速プロビジョニング	自動スケーラビリティ	自動冗長化
Rehost（リホスト）	×	△	△	△	△
Refactor（リファクタ）	×	△	○	△	○
Rearchitect（リアーキテクト）	△	○	○	○	○
Rebuild（リビルド）	○	○	○	○	○
Replace（リプレース）	―	―	―	―	―

○ 容易に対応可能　△工夫次第で対応可能　× 対応困難　― 利用するSaaSに依存

出典：ITR

クラウド・アセスメントの重要性と進め方

国内における企業のクラウド活用は進んでいるが、自社のクラウド活用方針を明確に定めている企業は多くない。アプリケーションの新規構築や再構築といった、特定の案件のために要件を定義し、複数のベンダーから入手した提案の総合評価からベンダーを選択した結果、ベンダー推奨のクラウド活用方法が選択されることが多い。その結果、自社の活用方針が不明確なまま、多種多様なクラウドが採用されていることが一般的である。このままでは、メインフレームからオープン化が進行した際にサイロ化に陥った過去の歴史を、クラウド時代にまた繰り返すことになろう。サイロ化が進行してIT運用／保守コストが肥大化し、ユーザーサイドからの期待や要望に応えられなくなってしまった過去を忘れてはならない。

自社のビジネスにおいてどのような方針や手法でクラウド・コンピューティングを活用するのか、という点で明確な戦略を持っている国内企業は少ないが、その多くは自社のクラウド利用での将来像は描いている。ITRが2019年4月に行った調査によると、「クラウド」と「オンプレミス」を併用する「ハイブリッドクラウド」を指向する企業が最も多く、次いで複数のクラウドを利用する「マルチクラウド」、単独のクラウドに集約する「シングルクラウド」の順となった。このいずれを指向する場合でも、クラウド上でのアプリケーションをどのように実装するのか、既存アプリケーションをどのように移行するのか、つまり「クラウド・アセスメント」に対する方針

は自社で決定しておく必要がある。ハイブリッドクラウドやマルチクラウドを指向する企業は、場当たり的にベンダーを選択して「クラウド・サイロ化」に陥ることのないよう、ベンダー選定基準を作成しておくことが必要である。本稿では「クラウド・アセスメント」について解説することとしたい。

ITRは、日本にクラウド事業者が登場する以前より、ビジネスにおいてクラウドが重要な役割を果たすと主張し、多数の企業にクラウド活用のコンサルティング支援を行ってきた。本稿で紹介するアセスメント・プロセスは、ITRがこれらのコンサルティング経験から理解しやすく構成した簡易版であり、全ての企業に万能とはいえないが、これを参考に自社にとって最適のアセスメント・プロセスを策定するとよいだろう。まずは、前述の2つのクラウド活用パターンのうちの既存アプリケーションのケースを解説する。既存アプリケーションに対してクラウドを活用するきっかけの例を図7に示した。

図7　既存アプリケーションのクラウド活用のきっかけの例

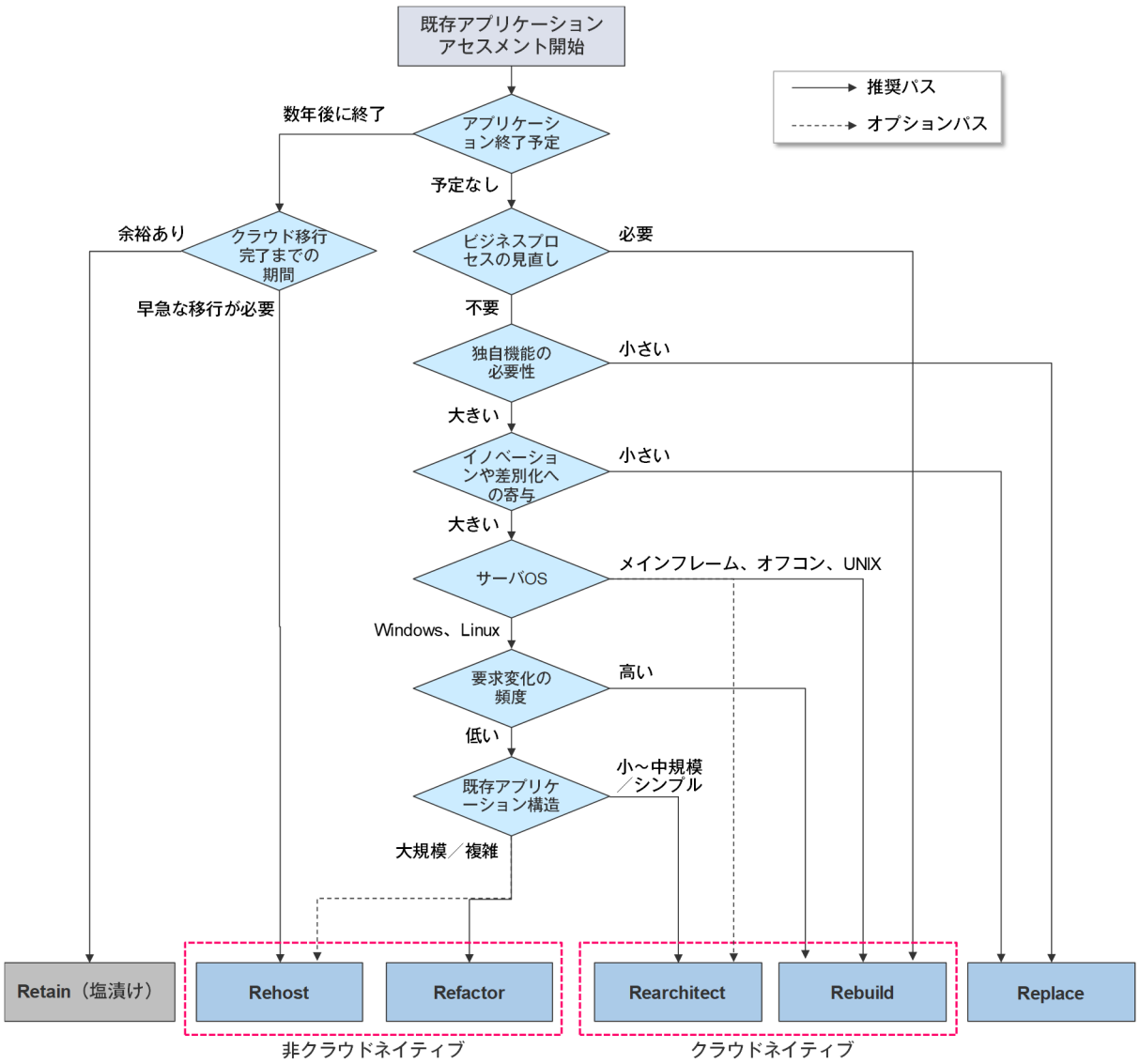
- 自社クラウド戦略に則った包括的なマイグレーション
- ソフトウェア／ハードウェアのサポート切れ
- 販売／開発した会社／部門の消滅
- 現行データセンターの契約切れ／消滅
- 企業合併
- ユーザー要件との乖離
  - ✓ 高パフォーマンス
  - ✓ 要求変化への適応
  - ✓ 伸縮性（需要への迅速な適応）
  - ✓ リソース拡大（データ増など）
  - ✓ グローバル展開
  - ✓ 高可用性
  - ✓ セキュリティ
  - ✓ コンプライアンス
  - ✓ 無停止運用

出典：ITR

既存アプリケーションの再構築

既存アプリケーションに対するアセスメント・プロセスを図8に示した。近いうちにアプリケーション利用を終了する予定があり、早急にクラウドに移行する必要がない場合は、現状のまま利用継続するのがよい（Retain：塩漬け）。現在利用しているデータセンターの終了期日が迫っている、あるいは全社方針で全アプリケーションをIaaSに早急に移行することが決定している場合は、Rehostすべきである。

図8　既存アプリケーション用アセスメント・プロセス



出典：ITR

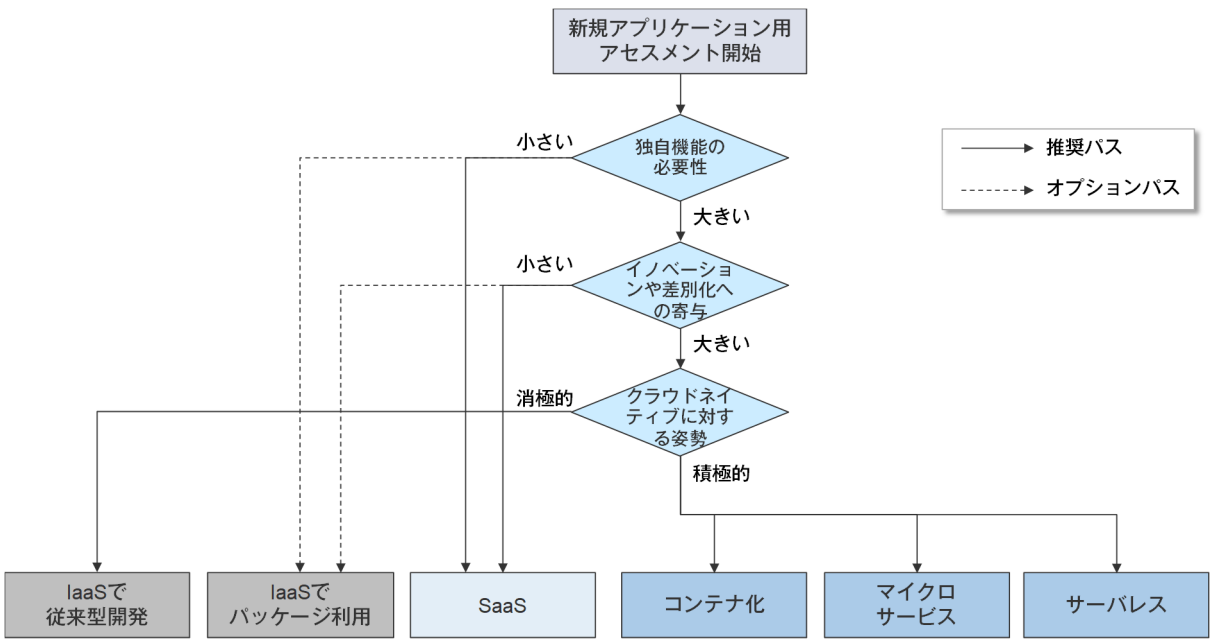
現時点で、アプリケーションの継続利用が決まっている場合は、入念なアセスメントが必要である。現行業務にさまざまな問題がありビジネスプロセスを見直す必要があるアプリケーションは、ゼロベースで再構築することが望ましい。ビジネスプロセスを見直す必要がなく、自社独自の機能が少ない場合は、その業務やアプリケーションに適合するSaaSを採用すべきである。自社独自の機能が多いが、イノベーションや他社との差別化への貢献度が小さいアプリケーションの場合は、その業務における独自機能を見直して、SaaSを選択するほうがよい。イノベーションや他社との差別化への貢献度が大きいアプリケーションで、サーバOSがメインフレーム、オフコン、UNIX（ここでは非IAアーキテクチャのサーバ・ハードウェアを指す）の場合は、現状のま

まクラウドに移行するのは非常に困難なため、アプリケーションの構造にもよるが、RebuildまたはRearchitectが望まれる。サーバOSがWindowsまたはLinuxで、アプリケーション・ユーザーの要求変化の頻度が高い場合は、現在のアーキテクチャではその変化に耐えられなくなる可能性が高いため、Rebuildすべきである。要求変化の頻度が低く、アプリケーションの構造が小～中規模または比較的シンプルな場合は、Rearchitectによりクラウドネイティブ化を目指すとよい。アプリケーションの構造が大規模または複雑な場合は、Rearchitectは多大な時間と費用を要すると考えられるため、Refactorでクラウドの利点を少しでも活かせる構造に変えるか、RehostでIaaSへの単純移行を採用することが望ましい。

新規アプリケーションの構築

新規構築におけるアセスメント・プロセスを図9に示した。自社独自の機能が少ない場合やイノベーションや他社との差別化への貢献度が小さい場合は、SaaSを選択すべきである。適切なSaaSが見つからず、パッケージ・ソフトウェアは存在する場合は、IaaS上でパッケージを利用するのもよいだろう。自社でクラウドネイティブ・アプリケーションの価値をまだ評価できておらず消極的な企業は、IaaSの仮想サーバを利用して従来通りの開発を行えばよいだろう。

図9　新規アプリケーション用アセスメント・プロセス



出典：ITR

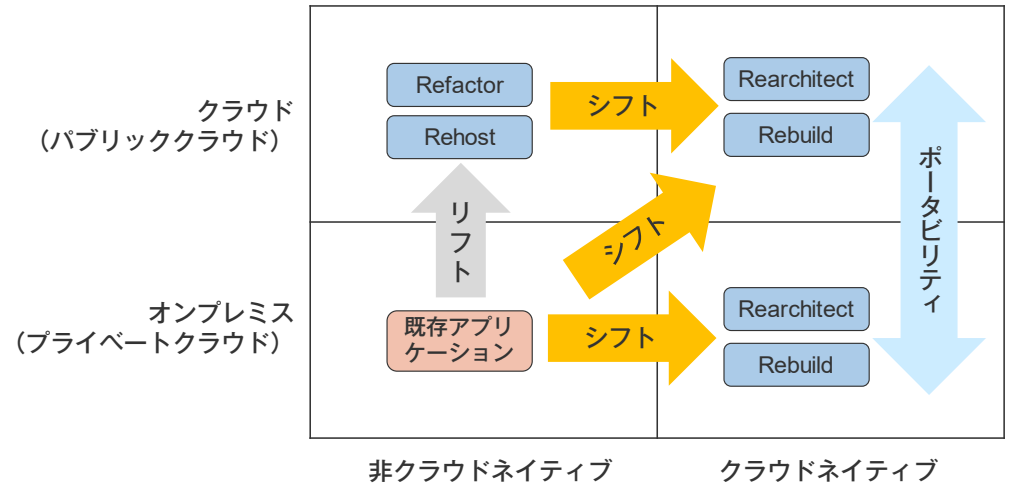
クラウドネイティブ・アプリケーションに積極的な企業は、コンテナ、マイクロサービス、サーバレスを活用して開発すればよい。クラウド活用経験が少ない企業ではマイクロサービスやサーバレスの活用を避けるケースもあるが、外部ベンダーの協力などを得てスモールスタートで開始することを推奨する。マイクロサービスやサーバレスの価値は自ら体験しなければなかなか理解できないためである。

リフトおよびシフトの概要

クラウドの世界では「リフト＆シフト」という言葉がよく用いられる。「リフト」とは、既存アプリケーションをそのままIaaSに移行する形態（Rehost）、または最小限の改修を行ってクラウドに移行する形態（Refactor）のことである。「シフト」は、非クラウドネイティブからクラウドネイティブへの転換を指し、RearchitectおよびRebuildに相当する。リフト／シフトの概念図を図10に示した。

前述の通り、オンプレミスで稼働すべきものもあるため、全ての既存アプリケーションをクラウドに移行する必要はない。しかし、シフトを検討する際は、クラウドネイティブ・アプリケーションのポータビリティ（クラウドおよびオンプレミスで同等に稼働すること）を担保しておくことが望ましい。オンプレミスにある既存アプリケーションをオンプレミス上でクラウドネイティブ・アプリケーションにシフトするケースにおいても、将来クラウドに移行する必要が発生するかもしれないためである。

図10　リフト／シフトの概念図



出典：ITR

第4章　リフト：既存アプリケーションのクラウド移行

クラウドネイティブ化をせずに既存アプリケーションをクラウドに移行する「リフト」は、直接クラウドネイティブ化する「シフト」に比べて価値が低いといわれることが多いが、そうではない。クラウドネイティブか否かにかかわらず、多くの既存アプリケーションをクラウドに移行することにより、ITインフラを標準化でき、統合運用や自動化が可能になり、グローバル展開も容易になるため、リフトによっても大きなメリットを獲得できる。

リフト手法の概要

既存アプリケーションを非クラウドネイティブの状態で移行する「リフト」には、さまざまな手法がある。以下に代表的な手法を示す。

- 手動移行
- 仮想マシン・イメージによる移行
- 移行ツール／サービスの活用

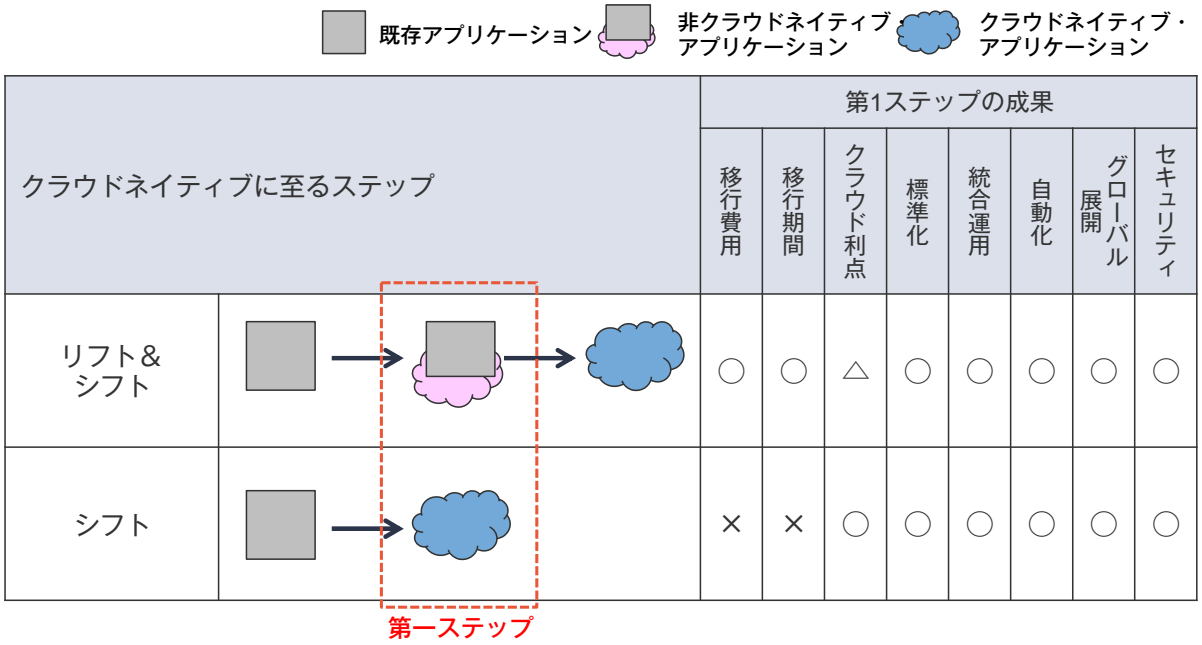
「手動移行」は事前に設計した手順に従って、サーバOS、ミドルウェア、実行環境、アプリケーション・コード、データなどを手作業で移行する手法である。SIベンダーが構築／運用／保守に関与したアプリケーションには、ITインフラ環境およびアプリケーション・デプロイのための手順書が作成されていることが一般的である。これらのドキュメントを利用して、既存アプリケーションのITインフラ上で行ってきた作業をクラウド上で行うのである。構成管理ツールやシェル・スクリプトなどを使って作業の効率化を図ることも多い。

多くのクラウド事業者は、VMwareなどのオンプレミス環境における仮想マシン・イメージをインポートするサービスを提供している。単独の仮想マシン上で稼働しているアプリケーションのリフトに最も用いられている手法である。また、一部のクラウド事業者は、複数の仮想マシン、ストレージ、ネットワーク機器で構成された複雑なアプリケーションの現行環境を分析して、自社クラウドにリフトするサービスを提供している。複数のデータセンターやクラウドに分散されている極めて複雑なアプリケーションを、単一のクラウドにリフトするサービスも存在する。

リフトの価値

一部のコンサルタントやITベンダーは、リフトはクラウドの利点を活かせないため価値がない、と説明しているが本当にそうであろうか。既存アプリケーションをクラウドに移行する典型的なパターンを図11に示した。「リフト＆シフト」パターンは、リフトを行った後にネイティブ・アプリケーションに改修または再構築する方法であり、「シフト」は、既存アプリケーションを直接クラウドネイティブ化する方法である。

図11　リフトの価値



クラウドネイティブに至るステップ		第1ステップの成果							
		移行費用	移行期間	クラウド利点	標準化	統合運用	自動化	グローバル展開	セキュリティ
リフト＆シフト		○	○	△	○	○	○	○	○
シフト		×	×	○	○	○	○	○	○

出典：ITR

既存アプリケーションをクラウドネイティブ化するには時間とコストがかかる。アプリケーションを数多く抱えている企業は、利用部門や開発ベンダーなどの関係者と調整しながら徐々に移行するとよいだろう。クラウド移行後のメリットとしては、図に示すように「リフト＆シフト」パターンは、「クラウド利点」の点では「シフト」パターンに劣るが、「移行費用」「移行期間」の点で優れており、「標準化」「統合運用」「自動化」「グローバル展開」という点は「シフト」パターンと同様に活かせる。このように、既存アプリケーションを即ネイティブ化しないリフトであっても、十分なメリットを獲得することが可能である。

思い通りにリフトが進まない企業

事前に計画したリフト作業をスケジュールおよび予算通りに遂行できた企業は少ない。企業の多くは、自社（IT子会社を含む）でアプリケーション開発／運用／保守を行っておらず、SIベンダーに頼っている。リフトを行う場合、必然的にSIベンダーに設計／実行／サポートを依頼することとなるが、SIベンダーは自らが作成したアプリケーション運用／保守のための手順書を流用するほうが確実であるし、新たなスキルを身につける必要がないため、手動移行を選択する傾向にある。しかし、クラウドは常に仕様が変化しているため、最初に作った手順書に随時修正が必要になることも少なくない。しかし、移行作業を担当するエンジニアは手順書を変更する権限が付与されておらず、変更のたびに設計者が分析や手順書の修正を行うことが多い。このような背景から、計画通りにリフトが進まないことが多いのである。

リフトの代表的な手法のひとつとして前述した「移行ツール／サービスの活用」は、そのツール／サービスに対する知識を得る必要があるため、ユーザー企業からSIベンダーに強く依頼しない限り、SIベンダーから主体的に提案が出てくることはないと考えたほうがよい。リフトにあたっては、複数の手法をユーザー企業のIT部門が調査し、比較評価を行い、自社が採用する手法を決定すべきである。

リフト事例

日本を代表するシステムインテグレーターのひとつである日本情報通信株式会社（以降、NI+C）では、オフィスへの入退出者管理に静脈認証アプリケーションを利用している。静脈認証端末はドアの裏表2台を1セットにして取り付けられており、利用者が入退室する際に指静脈認証を行っている。全ての入退出は静脈認証サーバによりトラッキングされている。静脈認証サーバが利用していたVMwareベースのITインフラ環境が今後リソースを増強しないことが決定され、新しいITインフラ環境に移行することが必要であった。

移行先としての選択肢は、別のVMwareベースのクラウドサービス、VMwareが利用可能なベアメタルサーバを提供するクラウドサービス、自社オンプレミス環境、仮想マシン構成を再構築し各クラウドサービスに移行、VMwareからの移行ツール「Migrate for Compute Engine」を提供しているGoogle Cloud Platform（以降、GCP）

があった。移行作業そのものにはビジネス価値がないため、移行コストを極力下げることが要件であった。また、今後多くの既存アプリケーションを新しいITインフラに移行する必要があるため、効率的に移行ができるツールの検討も必要であった。

これらの選択肢を評価した結果、クラウド事業で世界的実績があり、移行ツールが充実しているGCPを選定した。GCPが提供する移行ツールは下記の機能を備えており、今後多くのアプリケーションを移行する予定のNI+Cのニーズに合致するものであった。

- 移行前テストモード
- ストリーミングデータ転送

「移行前テストモード」は、GCP上のコンピュートリソース（CPU、メモリ、ネットワーク環境）を、オンプレミスVMware上の仮想サーバイメージ（ディスク）を利用して起動し、アプリケーション稼働確認を行う機能である。移行後のアプリケーション稼働に関しては、技術的な事前検証を済ませていることから、多くの場合問題なく稼働する。しかし、本番環境からの移行においては、まれにCPUタイプ、ソフトウェア・ライセンス、デバイスドライバ、ネットワークなどの違いによって正常に稼働しないこともある。移行直前に本番環境の移行後の動作確認ができる点が優れているとNI+Cでは判断した。

「ストリーミングデータ転送」は、ディスクの内容をクラウドストレージに直接転送することができる機能で、データ転送が完了していない状態は、スナップショットを利用してGCP上で仮想サーバを起動し、転送完了後切り替えることができる。この機能を利用することで複数仮想マシン構成のアプリケーションや転送に多くの時間がかかる場合でも、既存アプリケーションの影響を最小限に抑えつつ移行を行うことができる。

新アプリケーションは、サーバ側はGCPのIaaSであるGoogle Compute Engine（以降、GCE）、クライアント（静脈認証端末）側はオンプレミス機器とする構成とした。移行のための環境を図12に示す。

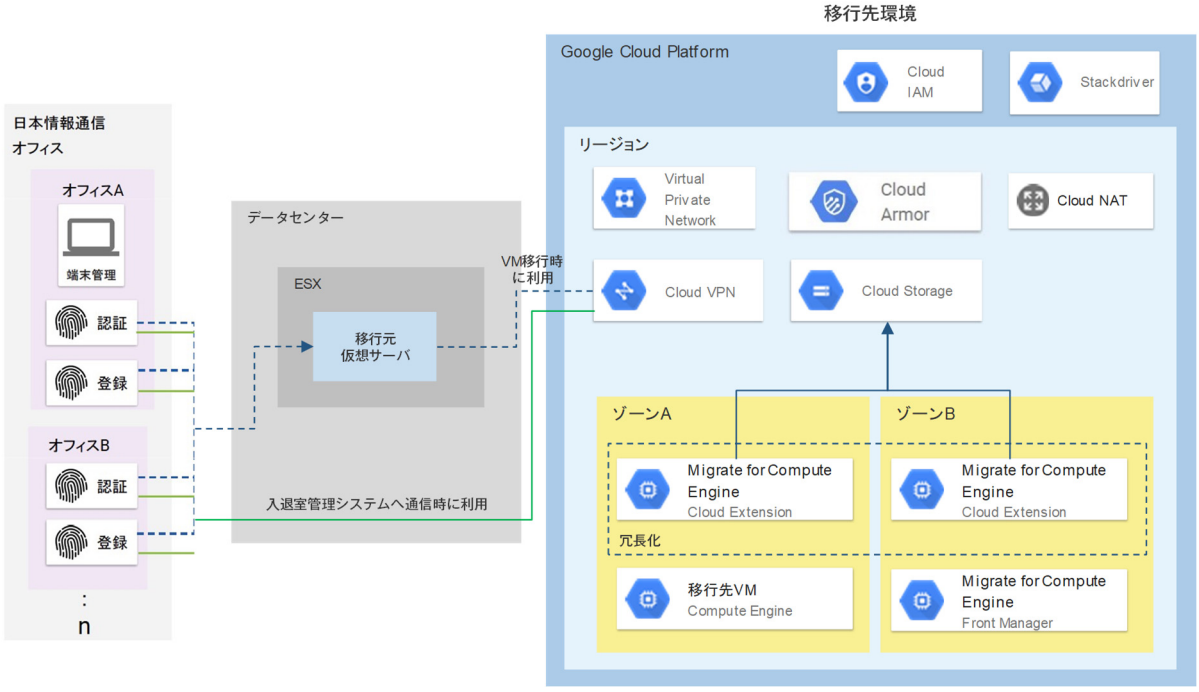
移行手順は下記の通りであった。

1. GCPとESXサーバ間のネットワーク確立
2. GCP上で、Migrate for Compute Engineサービスを利用するための管理サーバを

起動、ESXサーバ上でバックエンドサーバを起動し双方を接続し、Migrate for Compute Engineを構成

3. 移行時に仮想ストレージあるいはキャッシュとして利用されるCloud Extension ServerをGCEとして起動
4. VMwareの管理コンソール上から移行元の仮想サーバに対して「テスト起動」を実行
5. 起動後のGCP上の仮想サーバへの通信を確認。アプリケーションレベルでの動作に問題がない場合には「クラウド起動」を行い、本移行を完了

図12　日本情報通信におけるマイグレーション環境概要



出典：日本情報通信

移行作業はMigrate for Compute Engineの機能によって極めてシンプルに実行することができた。オンプレミス間での移行では、ストレージコピー時にアプリケーションを停止せざるを得ず、業務停止を余儀なくされていた。GCPのストリーミングデータ転送は業務停止時間を極小化できることがわかった。

以下の対策により、運用コストを約2割削減することができた。

- GCP移行時にITリソースを最適化した
- GCPから提供される各種サービス（リソース監視、ログ監視、スナップショット

など）の採用

- 従来、人的作業が必要だった作業をGCPから提供されるサービスに代替

これに加え、従来の監視サービスで取得していた以上のデータをGCPで取得でき、バックアップのタイミングを柔軟に調整できることから、より安全なアプリケーション運用ができるとNI+Cでは考えている。

第5章　シフト：クラウドネイティブ実現方法

アプリケーションをクラウドネイティブ化する、「シフト」のための重要テクノロジとして、マイクロサービス・アーキテクチャ、DevOps、Kubernetes、サーバレスがあげられる。いずれもこれらを採用すればクラウドネイティブ・アプリケーションが実現するというものではないが、効率的な開発／運用のために、これらのテクノロジを理解したうえで活用することが望まれる。

クラウドネイティブ・アプリケーションの構築手法

現時点でのクラウドネイティブ・アプリケーションを構築するためのキーワードを下記に示した。

- マイクロサービス・アーキテクチャ
- DevOps
- Kubernetes
- サーバレス

図13にクラウドネイティブ・アプリケーション実現のための各種手法をあげて、前述したクラウドネイティブ・アプリケーションの要件に対する充足度を比較した。

図13　クラウドネイティブ・アプリケーションの実現手法比較

利用テクノロジ／サービス	クラウドネイティブ・アプリケーションの要件				
	マイクロモジュール構造	迅速インテグレーション／デプロイ	迅速プロビジョニング	自動スケーラビリティ	自動冗長化
マイクロサービス・アーキテクチャ（MSA）	○	×	×	×	×
MSA＋DevOps	○	○	×	×	×
Kubernetes	×	×	×	×	○
Kubernetes＋MSA	○	×	○	○	○
Kubernetes＋MSA＋DevOps	○	○	○	○	○
サーバレス	×	×	○	○	○
サーバレス＋MSA	○	×	○	○	○
サーバレス＋MSA＋DevOps	○	○	○	○	○

○ 対応可能　× 対応困難

出典：ITR

マイクロサービス・アーキテクチャとクラウドネイティブ

マイクロサービス・アーキテクチャ（以降、「MSA」と略す）は、従来の巨大で一枚岩のアプリケーション（「モノリス」または「モノリシック」と呼ぶ）」が抱える問題点を解決するための重要な考え方である。適度な粒度で構成されるサービス群とサービス間連携にAPIを利用することが特徴であるこのアーキテクチャには、数多くの利点がある。個々のサービスは他サービスの内部プログラムやビジネスロジックを知る必要がなく、必要な情報をAPI経由で問い合わせ、情報を入手すればよい。また、個々のサービスは他のサービスから完全に独立しており、APIの仕様を変えない限り、サービス内のプログラム変更やロジック変更は他のサービスに影響しない。小人数で構成される開発チームで、サービス内のビジネスプロセス、UX（ユーザーエクスペリエンス）、データベーススキーマ、ビルド／デプロイなどのアプリケーション構築に必要な上流から下流までの全ての領域に責任を持つことが可能であるため、ビジネスを理解しながらアプリケーション構築／運用／保守を行うことも容易である。

しかし、MSAに基づきアプリケーションを構築したからといって、クラウドネイティブ・アプリケーションが実現するわけではない。MSAアプリケーションを構成する全てのサービスに対し、特別な仕掛けを施さない限り、図13に示した「迅速インテグレーション／デプロイ」「迅速プロビジョニング」「自動スケーラビリティ」「自動冗長化」を実現することは困難である。MSAアプリケーションとDevOpsを組み合わせれば「迅速インテグレーション／デプロイ」は実現するが、その他の要件は満たせず、クラウドネイティブ化は容易には実現できない。

Kubernetes＋MSA＋DevOpsの重要性

アプリケーションをコンテナ化して、Kubernetesによる運用を行うだけではクラウドネイティブ・アプリケーションは実現しない。Kubernetesは冗長化機能を内包しているので「自動冗長化」は容易に実現できる。Kubernetesは、コンテナランタイム管理、分散制御、コンテナ実行管理の機能に加え、複数のクラウド上のコンテナを統合管理する機能を有している。MSAと組み合わせれば、「迅速インテグレーション／デプロイ」以外の要件は実現可能である。KubernetesとMSAとDevOpsを組み合わせれば、クラウドネイティブ・アプリケーションを実現することができる。

クラウドネイティブ・アプリケーションを世界規模で推進しているCNCF（Cloud Native Computing Foundation）が提示しているクラウドネイティブ・アプリケーション実現のためのロードマップによると、コンテナ化が第一段階であり、次にDevOpsであるとしている。つまり、クラウドネイティブ化のためのコンテナ／KubernetesとDevOpsの組み合わせは非常に重要である。

サーバレス＋MSA＋DevOpsの重要性

サーバレスを利用すれば、ITインフラの運用／保守から完全に開放されて自社に必要なロジックをコーディングするだけでアプリケーション開発が可能となる。しかし、サーバレスを利用したアプリケーション構築を行ったからといって、自動的にクラウドネイティブ・アプリケーションが実現できるわけではない。MSAに基づいてサーバレスを活用すれば、「迅速インテグレーション／デプロイ」以外の要件は実現可能である。サーバレスとMSAとDevOpsを組み合わせれば、クラウドネイティブ・アプリケーションを実現することができる。前ページのKubernetesの項でも述べたように、クラウドネイティブ・アプリケーション開発において、DevOpsは非常に重要な役割を果たすのである。

既存アプリケーションのクラウドネイティブ化事例

富士フイルムグループの製品・商用サービス向けのソフトウェア開発およびITインフラ構築・運用を事業とする富士フイルムソフトウエア株式会社は、富士フイルムのプリントサービスやギフト販売を行っているネットショップ「FUJIFILMプリント&ギフト」のバックエンドを担う「オーダー管理システム（以下、OMS）」を2018年7月に刷新した。元のアプリケーションは構築後10年以上経過し、アーキテクチャが老朽化しており、下記の問題があった。

- 機能追加や修正を行う際の影響範囲の見極めが難しく、品質担保が困難で、保守コストが高い
- ユーザーアクセスの増加に伴ってリソースを増設する際の設計や事前検証に時間がかかり、また季節イベントやキャンペーン時に頻発するスパイク需要に対応するために余裕を持ったリソース構成にしていることから、運用コストが高い

ネットショップのフロントエンドはアプリケーション要件が頻繁に変化するため、OMSはそれらに柔軟かつ迅速に適応することが必要であった。またアプリケーション負荷に応じてリソースが柔軟に変動できるインフラも必要であった。本プロジェクトをリードした富士フイルムソフトウエアの小林大助氏（ソフトウェア開発本部イメージングソリューショングループ研究員）は、「従来アプリケーションの根本問題はオンプレミス上のモノリシック構造にある」と分析した。新アプリケーションは、クラウド上でマイクロサービス・アーキテクチャのアプリケーションを構築する方針で、コンテナを中心技術として採用することとした。再構築の検討を始める頃にはすでに小林氏はコンテナの可能性について調査をしており、コンテナの有望性を十分に理解していた。当時はKubernetesのマネージドサービスはどこからも提供されていなかったため、自力で全てインフラを構築・運用する必要があり、導入のハードルが高く諦めていた。しかし2017年末よりクラウドベンダー各社よりマネージドサービス提供が発表された。技術検証を行った2018年初頭に、商用のKubernetesマネージドサービスを提供しているのはGoogle Kubernetes Engine（以降、GKE）だけであったため、GCPを採用することとした。

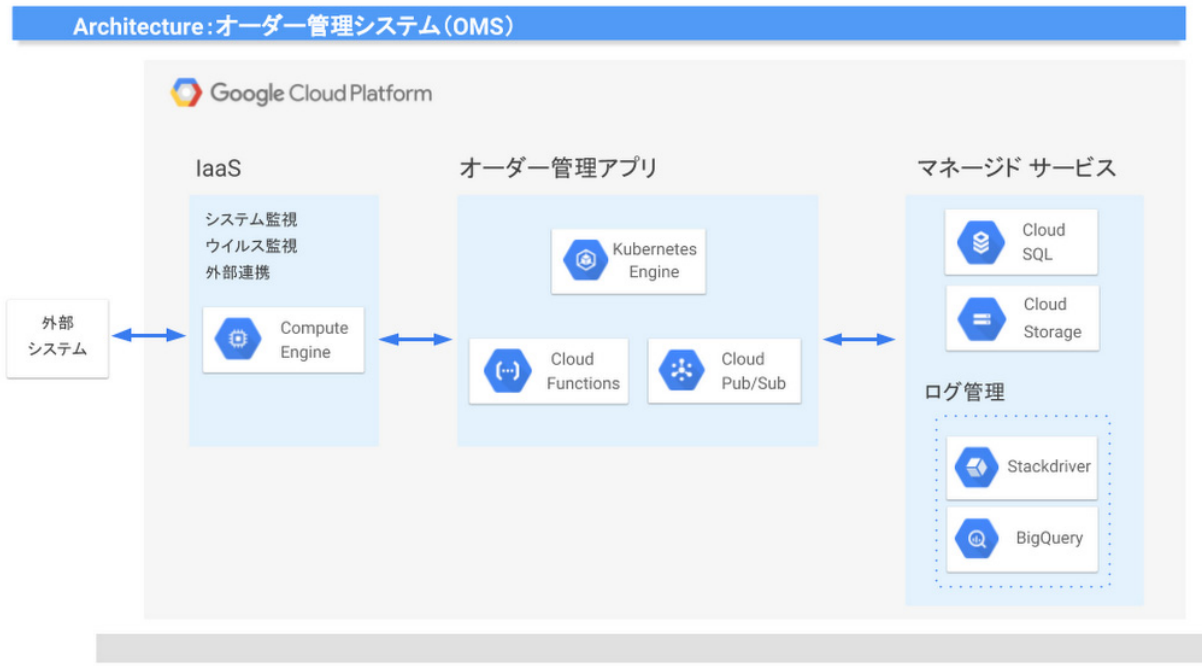
図14に新アプリケーションのシステム構成を示す。コンテナ・オーケストレーターのマネージドサービスGKEを中心として、ストレージはGoogle Cloud Storage、ログ分析／監視ツールはGoogle StackdriverとGoogle BigQueryを組み合わせ、データベースとしてGoogle Cloud SQL、メッセージングにGoogle Cloud Pub/Sub、既存アプリケーションとの連携にGoogle Cloud Functionsを採用した。基本的にマネージドサービスで構成されているが、IaaSでしか実現できない機能はGCEを採用した。このような設計により、運用／保守コストは大幅に削減された。マイクロサービス・アーキテクチャを採用したことにより、機能追加や修正時の影響範囲が限定され、開発の生産性も大幅に向上した。

事前に決定した新アプリケーションの稼働予定日を守ることが必須であった本プロジェクトでは、富士フイルムソフトウエア内でこれまで実績のないマイクロサービスやコンテナを活用することについて合意を得るのは容易ではなかった。コンセプトに対する異論は出なかったが、技術的な問題はないか、納期は守れるのか、といった心配が多くの関係者から寄せられた。小林氏は合意を得るための近道はないと考え、なぜこれらのテクノロジを採用する必要があるのか、これらを採用しなかった場合のリスクは何か、を理解してもらうため、資料を作成して関係者に何度も説明を行った。同社には先進テクノロジを調査研究評価する技術本部基盤技術グループという組織

があり、そこからの応援があったことも大きなポイントであった。

苦労した甲斐があり、関係者の合意が得られた後は、GKE中心に開発が進められた。GKEの学習コストは事前の予想よりも低く、検証も容易であった。アプリケーションおよびインフラ運用の手間が大幅に削減されたため、開発者がアプリケーション開発に集中できる環境を整備することができた。

図14　新オーダー管理システム（OMS）のアプリケーション概略構成



出典：富士フイルムソフトウエア

クラウドネイティブ・アプリケーションの新規構築事例

デジタル広告の世界においては、広告主がアプローチしたい顧客に対して最適なコストで広告を配信する仕組みである「デマンドサイドプラットフォーム（DSP）」や常に広告入札を行いターゲットユーザーに最も高い入札者の広告が表示される仕組みである「リアルタイムビッティング（RTB）」が主流となっている。株式会社フリークアウトは、日本で初めてこれらのサービスを開始した最大手事業者である。

同社が2017年9月にリリースしたデジタルメディア向けアドプラットフォーム・サービス「Red for Publishers」は、アドプラットフォーム構築、広告販売、オペレー

ション、コンサルティングを一気通貫で支援するものであり、システムのインフラストラクチャにはGoogle社が提供するクラウドサービスGCPのGKEを採用した。

フリークアウトにとってITはビジネスそのものであり、従業員の約5割がアプリケーション開発者で、常に最新テクノロジを活用してサービスを構築してきた。取締役CTOである西口次郎氏は、かねてよりコンテナおよびKubernetesを活用する機会を探索していたが、「Red for Publishers」の企画が持ち上がった時に、GCP東京リージョンが公開されていたこともあり、GKEを利用しようと考えた。KubernetesはGoogle社において商用として長年使用された実績があることから、Google社が提供しているKubernetesのフルマネージド・サービスの成熟度や信頼性は高いと考えたためである。デジタル広告においてレスポンスは極めて重要な要素であるため、データセンターが東京にあることは必須要件であった。また、同社はグループ企業の広告プロダクトのログ基盤にすでにGCPのBigQueryを採用しており、その高い柔軟性、自動スケール機能、大規模システムにおけるパフォーマンスと安定性を高く評価していた。

フリークアウトでは最初からクラウドサービスを利用していたわけではない。前述の通り、デジタル広告においてはレスポンスが肝要であるため、元々はオンプレミスシステムが主流であった。オンプレミスシステムは、カスタマイズ性や安定性の高さ、コンピューティングリソースの一括確保が可能というメリットはあるものの、ハードウェアの調達や運用／管理に多くの時間を費やし、OS／ミドルウェアのパッチ適用などの運用作業が繁雑であった。KubernetesのマネージドサービスであるGKEを利用することにより、セキュリティ管理やデプロイ作業に時間を割くことがなくなり、ビジネスに直接寄与するシステム開発に集中できると考えた。

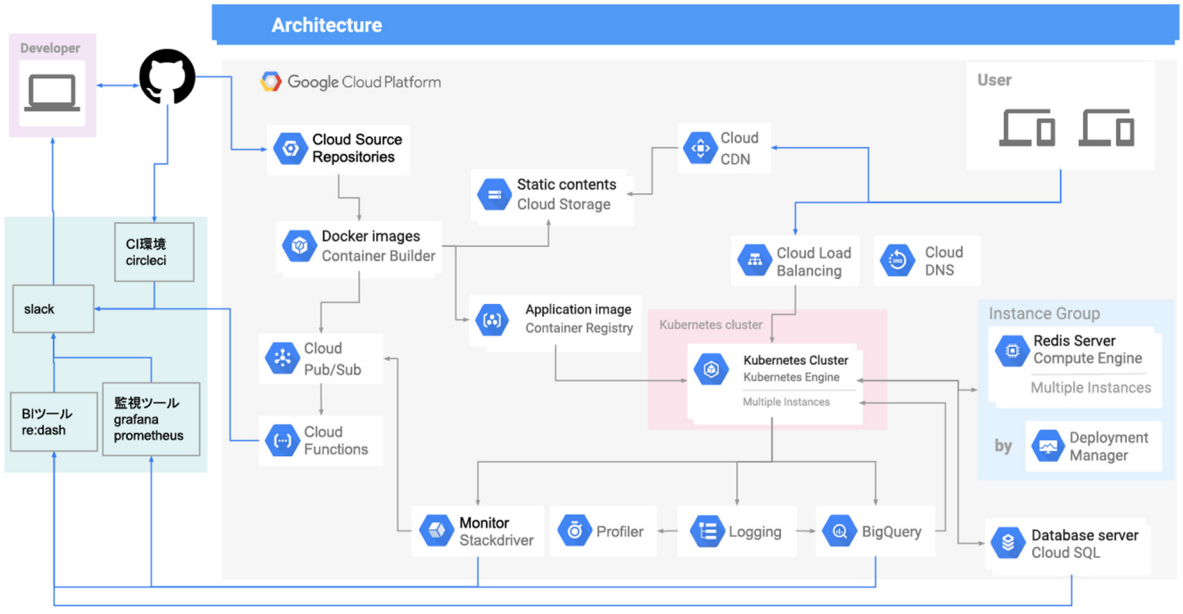
2017年4月に開発がスタートし、インフラストラクチャの設計／検証／実装は全てCTOである西口氏が1人で行った。本プロジェクトは同社にとって初めてのGCP本格導入であったが、Google社から提供されているドキュメントとソースコードを参照しながら、自力でシステムを開発し、問題なく7月には本稼働を開始した。

「Red for Publishers」のシステム構成を図15に示した。メインシステムは、GKE上で稼働している。Kubernetesのインスタンスは、可用性や検証のしやすさを考慮し、広告配信、広告管理、バッチジョブの3構成を採用している。Kubernetesのモニタリングにはgoogle Stackdriverを利用している。GCPは提供されているサービスのアッ

プデートや新規サービスの追加が頻繁に行われるため、追加コストが不要で機能拡充や利便性向上を図ることができる点が魅力であると同社は考えている。また、OSSを基盤としているため、ソースコードにアクセスできることもGKEを採用したポイントであった。デジタル広告用システムは負荷が非常に高く、かつ優れたパフォーマンスが要求されるため、その分トラブルも発生しやすい。同社では自社でトラブルシューティングを行うことから、ソース非公開の商用ソフトウェアは採用しない。

フリークアウトのようにCTO自らが実際にシステム企画から構築まで行うという企業は国内では珍しいといえよう。同社にとってITはビジネスそのものであるため、高いスキルを持つ技術者を採用し続ける必要があると考えている。レベルの高い技術者と一緒に働き続けるには、常に最新テクノロジを採用して、技術者に同社のシステム開発に携わることが面白いと感じてもらうことが必須である、と西口氏は話す。このような考え方やアプローチは、DXに取り組んでいる企業も大いに参考にすべきである。

図15　「Red for Publishers」のアプリケーション構成



出典：フリークアウト

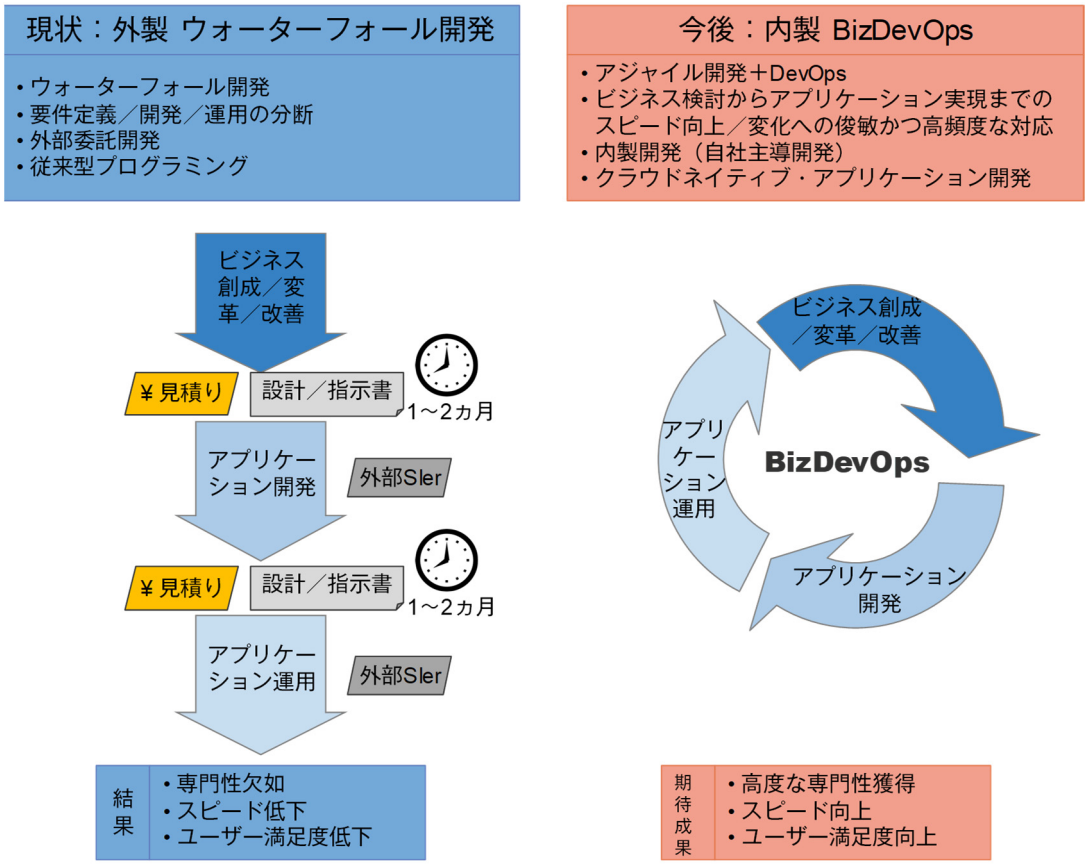
第6章　提言

かつてメインフレームが主流の時代は、企業は内製でアプリケーション開発を行っていた。この時代は、端末、OS、ミドルウェア、ネットワークはメインフレームベンダー独自のテクノロジを利用する企業が多く、ベンダーのカスタマーエンジニア（CE）がこれらの運用を担当し、IT部門は全社的に標準化されたCOBOLやPL/Iといったプログラム言語を使ったコーディングに専念できた。その後、メインフレームからオープンシステムの時代になり、クライアントデバイス、サーバOS、ミドルウェア、ネットワークが多様化し自由に選択できるようになったが、その反面、IT部門の業務範囲が飛躍的に拡大したために、企業の多くはシステム設計やコーディングを外部ベンダーに委託する（外製）方針を採った。インターネット時代になって、IT部門の守備範囲はセキュリティやコンプライアンスなどへさらに拡大し、ほとんどの企業では、外製化は当然であり、IT部門はユーザー要求を外部ベンダーに橋渡しする役割を担っていると捉えている。

現在多くの企業が採用しているアプリケーション開発プロセスを図16の左側に示した。ウォーターフォール開発に基づき、ユーザー部門と共同で要件定義を行い、複数のベンダーにシステム設計／開発の見積りを依頼する。ベンダーの提案を評価し開発ベンダーを選定した後は、IT部門はベンダーの進捗管理を行うに過ぎず、システムアーキテクチャ、開発言語、ミドルウェアはベンダー任せである企業が多い。システム開発が完了した後は運用のための要件をIT部門が策定し、ベンダーに運用設計／遂行の見積りを依頼し、ベンダーを選定する。このように段階ごとに大きな意思決定プロセスが挟まることから、要件定義／開発／運用の分断が発生し、スピードも柔軟性も欠けたものとなる。

また、このプロセスでは、アプリケーション開発経験に長けた技術者が要件定義の段階で参画しないため、先進テクノロジを活用したアプリケーション開発を企画することが難しくなっている。さらに、ベンダーが採用するテクノロジや開発手法に対する見極め力をIT部門が持ち合わせないことも多い。開発では、ベンダーは顧客企業が定義した要件だけで開発を行うことは非常に困難なため、工数を多めに見積り、高コストになっている。またウォーターフォール開発のため、開発途中の要求変化に対して柔軟に対応することができない。何より、アプリケーション開発経験やスキルが社内に蓄積されないことが極めて大きな問題である。自社アプリケーションの中身がわからず、機能追加や修正が可能かどうかわからないIT部門が多くなっている。

図16　DXのためのBizDevOps



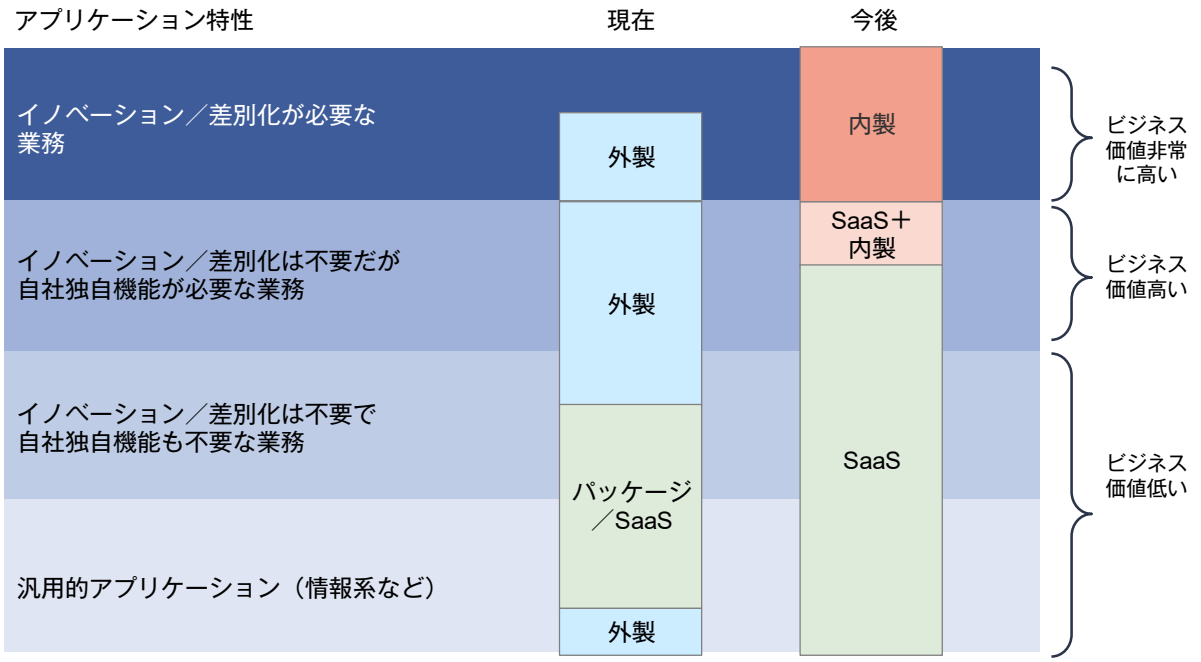
出典：ITR

DXに対する要望が高まっている現代においては、ビジネスアイデアを迅速にアプリケーション化し、それを利用した結果を早期にフィードバックしてより良いアプリケーションに仕上げていくことが重要である。そのためには、アジャイル開発とDevOpsを組み合わせ、「ビジネス創成／変革／改善」「アプリケーション開発」「アプリケーション運用（ユーザー利用）」を短期間で頻繁に繰り返すプロセスが必要となる。本稿では、このプロセスをBizDevOpsと呼ぶ。これを効率的に回すには、ベンダーに都度見積りを要求する従来のプロセスを適用せずに、内製化を推進すべきである。先進テクノロジやITを熟知した技術者が業務部門と共同でアプリケーションを開発することが望まれているのである。

企業が利用するアプリケーションは多岐にわたるため、全てのアプリケーションを内製化することは非現実的である。自社ビジネスを理解した自社要員による内製はイノベーションや差別化が必要な業務、つまりビジネス価値が高いアプリケーションに特化して適用すべきである（**図17**）。本稿の図8および図9で述べたアセスメント・プ

ロセスに基づいて内製を行うべき業務を決定し、クラウドネイティブ・アプリケーションを開発すべきである。業界特有の商習慣や差別化にはつながらないが自社固有のビジネスプロセスを撤廃することが困難な業務は、独自機能の必要性を精査したうえで、SaaSをベースにAPIを利用した独自開発アプリケーションを付加する方針にするとよい。

図17　内製対象のアプリケーション



出典：ITR

このようなプロセスに舵を切ることは決して容易ではない。しかし、今ではクラウドネイティブ・アプリケーション開発を試すことは容易で低コストである。ここで示したBizDevOps手法を小規模で試行し、その良さをIT部門自らが体感し、経営者やビジネス部門などの関係者に理解させることでクラウドネイティブ・アプリケーションの内製化が加速するであろう。

分析／執筆: 甲元 宏明
text by Hiroaki Kohmoto

ITR White Paper

「2025年の崖」から落ちないためのアプリケーション変革
～マイクロサービス、DevOps、コンテナを採用したクラウドネイティブへの道～

C19110126

発行　2019年11月14日

発行所　株式会社アイ・ティ・アール

〒160-0023
東京都新宿区西新宿3-8-3　新都心丸善ビル 3F
TEL：03-5304-1301（代）
FAX：03-5304-1320