アプリケーション開発運用手法の新たな潮流「マイクロサービス」

変化に強いクラウド・ネイティブ・アプリケーション開発運用の指針

ビジネスの成長を継続するには、取り巻く環境の変化を素早くつかみ、ビジネス・モデルのトランスフォーメーションを果敢に実践する積極性が重要です。クラウド・コンピューティングは、そのようなビジネス・トランスフォーメーションのニーズに応えるICT（Information and Communication Technology）のイノベーションです。クラウド・サービスの一つであるIaaS（Infrastructure as a Service）は、ICT基盤構築運用のスピード・アップと費用削減を実現してきました。そして今、より上位層のアプリケーション・プラットフォームのイノベーションとして、PaaS（Platform as a Service）に期待が集まっています。

とはいえ、PaaS上でこれまでと同じミドルウェアを用意し従来型のアプリケーションを動かしたのでは、その効果は限定されてしまいます。クラウドの能力を最大限に活用してビジネスをドライブするスピーディーなICTシステムを実現するには、クラウド・コンピューティングに見合ったアプリケーション開発運用手法を用いるべきです。そのようなアプリケーション開発運用手法の一つが「マイクロサービス」です。本稿では、今注目を集めているマイクロサービスについて概観します。

1. マイクロサービス登場の背景

ビジネスのグローバル化によって市場の垣根が取り払われ、消費者の嗜好も多様化しています。従来の商品やビジネス・モデルがあっという間に陳腐化してしまうことは珍しくありません。ビジネスを成長させるためには、消費者ニーズをつかみ、商品やサービスを素早く市場に投入することがますます重要になります。

このような中で、ICTシステムがお手伝いできる機会もまた増えてきています。ビッグデータとアナリティクスによる市場動向の把握は新たな市場創出に有効であり、インターネットとモバイルはビジネス取引から時間と場所の制約を取り払います。

ICTシステムは、ビジネスにおける役割が重要になるほど、拡張性とスピードが求められます。消費者ニーズの変化に応じて素早く商材を投入する必要があるため、ビジネス基盤を担うアプリケーションは機能変更と拡張を継続しなければなりません（アプリケーションのメンテナンス性）。また、消費者からのリクエスト数の増減に関わりなく安定して運用できることも重要です（スケーラビリティー）。いずれにしても、ビジネス・スピードに合ったタイムリーな対応が必要です。

基盤構築のスピード・アップや基盤のスケーラビリティーは、基盤の仮想化と継続的デリバリーによって実現可能です。しかし、アプリケーションのメンテナンス性やスケーラビリティーを講じるには、アプリケーションの設計思想や開発プロセスに踏み込んだ手法を実践する必要があります。そのための手法の一つが「マイクロサービス」（Microservices）です。

2. マイクロサービスの概要

マイクロサービスとは、クラウド基盤上で稼働するアプリケーションを開発・運用する上でのアーキテクチャー・スタイルで、ICTシステムの設計指針のみならず、プロ

ジェクト運営のプラクティスも含まれています。米国ThoughtWorks社のチーフ・サイエンティストであるマーティン・ファウラー氏が、同社コンサルタントが実際のクラウド・システム開発の現場で経験した知見をまとめ、自身のWebサイトで公開[1]したのを機に広く知られるようになりました。

マイクロサービスのゴールは、アプリケーションのメンテナンスをより容易にし、反復型開発を実現するところに置かれており、そのために次の基本方針を挙げています。

- ●小規模なサービスを組み合わせてアプリケーションを開発すること
- ●各サービスは、独立したプロセス上で稼働すること
- ●各サービスは、RESTなどの軽量プロトコルで通信すること
- ●各サービスは、自動的に、それぞれ個別にデプロイできること
- ●各サービスは、それぞれ異なるプログラミング言語で実装可能（Polyglot Programming[2]）であり、かつ、それぞれ異なる永続データストアを利用可能（Polyglot Persistence[3]）であること

言い換えれば、置き換え可能なサービス単位でアプリケーションを組み立てることで、アプリケーションのメンテナンス性を確保しようというのがマイクロサービスのエッセンスです。また、リクエスト数の多いサービスが稼働するプロセスのみをスケールすることで、限られたリソースの中でも効率的にリクエストの増大に応えることができます。さらに、PaaSや継続的デリバリーを利用してデプロイを自動化することで、スピード・アップも図れます。マイクロサービスは、アプリケーションのメンテナンス性、スケーラビリティー、そしてスピードに対する課題を解決するアプリケーション開発運用の手法なのです。

3. マイクロサービスの特徴

現実のクラウド・システム開発現場のフィードバックを基にした、いわば草の根から発生したマイクロサービスには、Java EEやWebサービスのような確立された仕様書はありません。それに代わるものとして、マーティン・ファウラー氏が挙げているマイクロサービスの9つの特徴が一種のガイドラインとして目されています（**図1**）。これらはマイクロサービスを実践する上での絶対条件というわけではなく、状況や要件に応じて適切な特徴を取り込んでサービスを開発し運用すれば良いとされています。以降では、マイクロサービスの特徴を簡単に解説します。

3.1. サービスによるコンポーネント設計

マイクロサービスはサービスによるコンポーネント化を推奨しています（**図1.Componentization via Service**）。その理由は、独立したプロセス上でホスティングされたサービスは容易に置き換え可能であり、

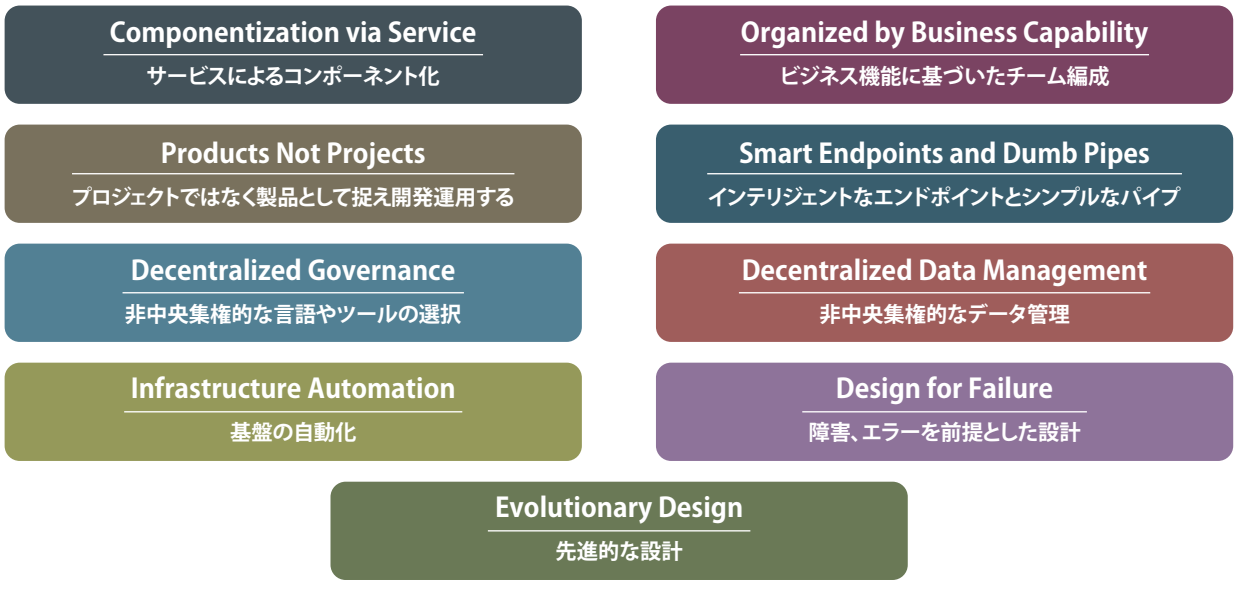

図1. マイクロサービスの9つの特徴

アプリケーションのメンテナンス性向上に寄与するとともに（**図1.Evolutionary Design**）、特定のサービスのスケーラビリティーを実現しやすいからです。非マイクロサービス・アーキテクチャー・スタイルであるモノリス（Monolith：一枚岩アーキテクチャー）の場合、一部のアプリケーション・ロジックの改修に際しても、アプリケーション全体のコンパイル、ビルド、テスト、デプロイが必要となりますが、マイクロサービスに基づいてアプリケーションをより細粒度のサービスに分割しておけば、改修対象のサービスについてのみ作業を行うだけで済みます（**図2**）。またモノリスは、特定のアプリケーションのみスケールさせるアーキテクチャーではありません[*1]。

ただし課題もあります。独立したプロセスでサービスが稼働するため、サービス間連携の都度プロセス間通信が発生し、パフォーマンスに悪影響を与える可能性があるのです。これを抑えるために、サービス粒度を粗くして、プロセス間通信の頻度を抑えるような配慮が必要です。

マイクロサービスでは、複数のサービスが協調して一つのリクエストを処理しますが、サービス間の通信には軽量でシンプルな通信手段を用います。RESTや、軽量なメッセージングがその例です（**図1.Smart Endpoints and Dumb Pipes**）。メッセージング・エンジンとしてEnterprise Service Bus（ESB）を利用することは構いませんが、メディエーションの利用は推奨されません。理由は、メディエーションの仕組みは複雑で使いこなすのが難しく、不具合の一因になりがちであるためです。

3.2. 開発運用体制

マイクロサービスでは、プロジェクトの実働部隊である各開発運用チームをそれぞれ小さな単位に抑えることが推奨されています。一つのサービスが責務を負うビジネス要求の単位で、開発運用チームを編成します（**図1. Organized by Business Capability**）。そして、独立して開発運用できるように、この小さなチームには、ユーザー・インターフェース、アプリケーション・サーバー（アプリケーション・ロジック）、DBそれぞれの専門家を投入します。この背景には、コンウェイの法則（Conway's law）があります。ICTシステムの構造は開発プロジェクト体制を反映する、というのがこの法則の骨子です。確かに、チームをユーザー・インターフェース、アプリケーション・サーバー、DBに分割すると、でき上がったICTシステムはそれぞれのチームの成果物が一つの層を成す、伝統的な3層構造となることが珍しくはありません。マイクロサービスの目標は、3層構造のWebアプリケーションを作り上げることではなく、独立してメンテナンス可能な比較的小さなサービスを作成することです。コンウェイの法則に従うと、一つのサービスの開発運用を、一

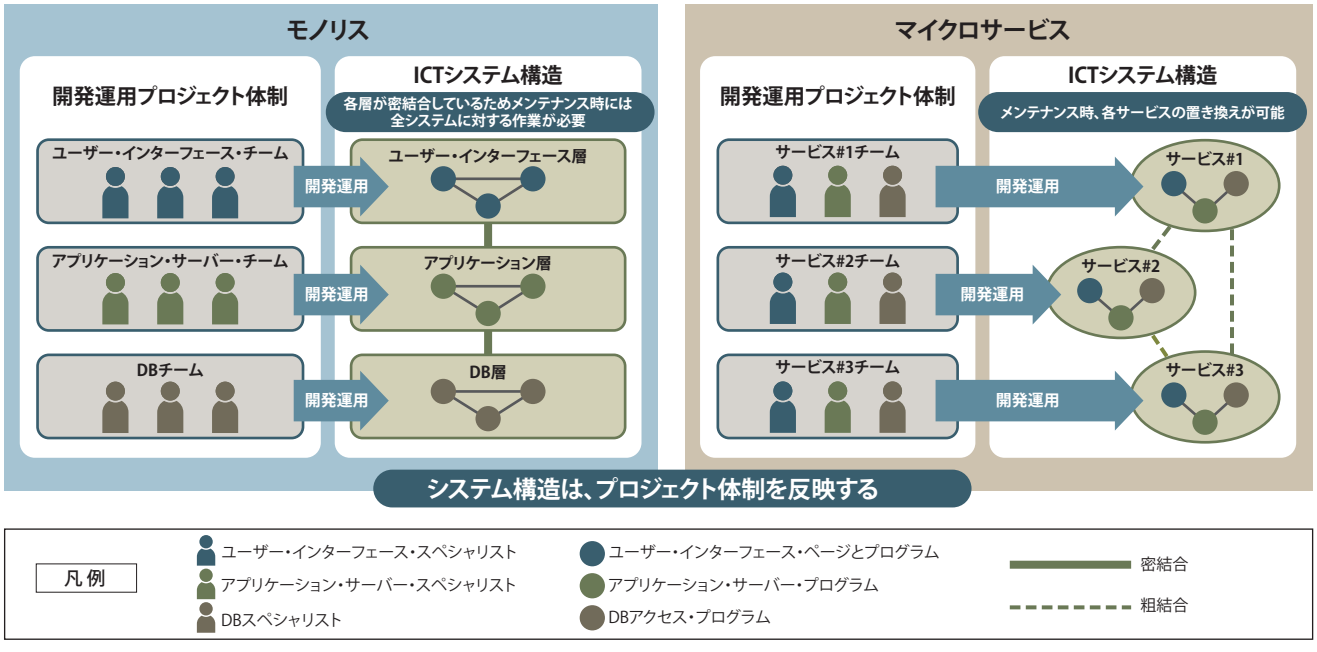

図2. マイクロサービスとモノリスの比較―構造と開発運用体制

つのチームが担うのは理にかなった判断です（**図2**）。

ここまで、プロジェクトの実働部隊の最小単位を“開発チーム”ではなく“開発運用チーム”と記したのは、意図があってのことです。ICTシステムを一つの製品と見なし、そのライフサイクル全般を一つのチームで面倒を見るのが、マイクロサービスの流儀です（**図1.Products Not Projects**）。開発者が、メンテナンスにもある程度責務を負うことで、エンドユーザーのフィードバックをシステムに確実に適用できるようになります。これは、反復型開発を実践する上で重要な指針と言えます。

3.3. 開発環境と永続データストアのガバナンス

ICTシステムの開発運用の現場において標準化は非常に重要であり、プログラミング言語、ソフトウェア製品の種類やバージョンなどを一意に規定しています。その一方で、マイクロサービスでは、プログラミング言語やツールなどの選定は各サービスの運用開発チームに委ねます（**図1.Decentralized Governance**）。各サービスがそれぞれ異なるプログラミング言語やツールを採用することを許容し、それが最適な選択であれば「GitHub」などインターネット上で共有されているツールも採用します。これができるのは、“Products Not Projects”という方針の下、チームがサービスのライフサイクル全般に責任を持っているからです。

同様に、永続データストアについての判断も各チームに任されています。各サービスが独立したDBインスタンスを利用したり、あるいはリレーショナルDBやキー・バリュー・ストアなど異なる形態のデータストアを利用したりすることも許容されます（**図1.Decentralized Data Management**）。マイクロサービスでは、データの用途や性格に応じて、最適なデータストアを利用することが推奨されているのです。

非中央集権的なガバナンスの結果として、サービスもデータも共に分散配備されることになりますが、どのようにデータの整合性を担保するのでしょうか。マイクロサービスは、X/Open XAやWS-Transactionなどの分散トランザクション・プロトコルの利用は推奨しません。その理由は、サービス間通信にESBのメディエーションを推奨しないのと同様に、複雑で不具合の要因になる可能性が高いためです。分散トランザクション・プロトコルによってデータのACID属性を保つ代わりに、マイクロサービスでは結果整合性（Eventual Consistency）モデルの採用を推奨しています。障害などによってデータに不整合が発生した場合、修復のためのコンペンセーション・オペレーションを講じるのです。

3.4. 基盤に対する考慮

マイクロサービスは、基盤環境構築、ソフトウェアのコンパイル、ビルド、テスト、デプロイの自動化を推奨します（**図1.Infrastructure Automation**）。素早い環境構築が可能となり、開発局面において十分なテストが可能となるので、システム品質向上に寄与します。また本番局面においては、オペレーション・ミスの少ない素早いメンテナンスを実現します。

ICTシステムには必ず不具合が生じます。そこでマイクロサービスは、障害発生を前提とした設計を提案しています。サービスごとに適切なタイミングで十分な内容のログを出力することはもちろんのこと、タイムリーに障害など不具合を検知する監視システムの構築と運用を推奨しています（**図1.Design for Failure**）。

4. マーケットの反応と教訓

サービス化という観点でしばしば比較されることが多いSOA（Service Oriented Architecture）とは異なり、マイクロサービスの原点はICTシステム開発の現場からのフィードバックです。期待や議論にとどまらず、事例も出てきています。その一つが「IBM Bluemix ユーザー・インターフェース」（Bluemix UI）です。

IBMのPaaSであるBluemixは、機能拡張のためほぼ週次でメンテナンスされています。高頻度のアプリケーション・メンテナンスが必要な大規模システムである点で、Bluemixはマイクロサービス化の対象として最適な例の一つと言えます。

Bluemix UIはエンドユーザーとの接点という重要な役割を担っていますが、週次メンテナンスのボトルネックとなっていました。Java EEベースのSingle Page Applicationとして実装されていた当時のBluemix UIは典型的なモノリス構造となっており、一部のアプリケーションの変更に際しても、Bluemix UI全体のコンパイル、ビルド、テスト、デプロイの必要があったためです。こ

の状況を改善するために、Bluemix UIはNode.jsを利用してマイクロサービス構造に書き換えられました [4] [5]。このコンバージョンによって、アプリケーション変更の影響を局所化することに成功し、Bluemixのメンテナンスは以前と比較して非常にスムーズに実施されています。

一方、マイクロサービスに対する慎重な意見も少なくありません。その代表的な意見の一つに、設計局面でのサービス切り出しの難しさがあります。SOAにおいても同様の指摘がありましたが、サービスの境界をどこで区切るか、言い換えればサービスの粒度はどの程度であるべきか、汎用的なガイドラインがないのです。結果として、サービスの品質にばらつきが生じ、コンポーネント化はしたもののメンテナンス性が伴わない、使いづらいアプリケーションに陥る恐れがあります。

この指摘に対するカウンター・クレームの一つとして、“モノリス・ファースト (Monolith First)”[6]というアプローチがあります。最初の開発時には、従来どおりモノリス構造でアプリケーションを設計実装し、その後の反復開発の都度、段階的にアプリケーションの一部をサービス化していくというのがそのコンセプトです。数回の反復開発を経ることで、アプリケーションの構造はモノリスからマイクロサービスに転換できるというわけです。ドメインのスペシャリストであっても、ただ一度の開発プロジェクトで完璧なサービスの境界を決めることはできないと言われています。反復開発を利用して段階的にサービス化を進め、洗練させるモノリス・ファーストは、現実的で理にかなった手法だと思います*2。

5. 終わりに

従来のアプリケーションをほぼそのまま仮想化環境に載せ替えることを“クラウド化”と呼んでいるケースは少なくありません。これは単純な基盤環境のリノベーションに過ぎず、得られるメリットは多くはありません。その理由は、アプリケーションのユースケースや開発手法については新たな試みがなく、投資効果が基盤構築の期間短縮とコスト削減に限定されるからです。

クラウド・コンピューティングのメリットを享受するには、アプリケーションも含めて新たな“ユースケース”と“手法”を実践しなければなりません。クラウド環境におけるアプリケーションの開発運用手法の新たな提案であるところにマイクロサービスの価値があります。

そのネーミングから“サービス化”に注目しがちですが、“反復型開発”にこそマイクロサービスの意義があると、筆者は考えます。ICTシステムがビジネスにより多くの責任を持つ現代、開発運用チームがICTシステムのライフサイクル全般をカバーし、フレキシブルにシステムをアップデートしながらビジネスをドライブする方向性が重要です。マイクロサービスによる“サービス化”は、そのような反復を促進する手段なのです。

*1：モノリス構造のアプリケーションであっても、特定の製品機能を用いることで、特定のアプリケーションのスケールを実現することは可能です。
*2：モノリス・ファーストには、ここで挙げた以外の手法もあります。また、モノリス・ファーストに対する反論もあります。

[参考文献]
[1] Fowler, Microservices, http://martinfowler.com/articles/microservices.html, 2014年3月25日
[2] Ford, Polyglot Programming,http://memeagora.blogspot.jp/2006/12/polyglot-programming.html, 2006年12月5日
[3] Fowler, “PolyglotPersistence”, http://martinfowler.com/bliki/PolyglotPersistence.html, 2011年11月16日
[4] Martin, Erwin, Migrating a Monolithic App to Microservices on Cloud Foundry, http://www.slideshare.net/Pivotal/cloud-foundry-summit-2015-migrating-a-monolithic-app-to-microservices-on-cloud-foundry, 2015年5月15日
[5] Martin, Erwin, Migrating a Monolithic App to Microservices on Cloud Foundry, https://www.youtube.com/watch?v=UauTkqWql_8, 2015年5月14日
[6] Fowler, MonolithFirst, http://martinfowler.com/bliki/MonolithFirst.html, 2015年6月3日

日本アイ・ビー・エム株式会社
クラウド事業統括, クラウド・テクニカル・ソフトウェア テクニカル・セールス

樽澤 広亨
Hiroyuki Tarusawa

IBMソフトウェア製品（WebSphere Application Serverおよびその派生製品）の、開発エンジニア、エバンジェリスト、テクニカル・セールスなどを経て現職。情報処理学会, 情報企画調査会, SC38専門委員会, 専門委員。