redhat

アジャイル・インテグレーション

エンタープライズ・アーキテクチャのブループリント

E ブック

Steve Willmott、David Codelli 共著
Deon Ballard 編集

目次

ビジネスを成功に導く要因として、変化への対応力の比重は右肩上がりで大きくなっています。破壊的革新をもたらすプレーヤーが市場に参入し、テクノロジーが消費者の期待を大きく左右する状況において、組織がこれまでよりもはるかに短いサイクルで計画を変更する必要性が高まっています。その中で、最新のソフトウェア・アーキテクチャとプロセスを取り入れた組織はこの変化に対応し、その市場の勝者になることができるでしょう。

アジャイル・インテグレーションと呼ばれる新しいアーキテクチャのフレームワークは、コンテナ、分散インテグレーション、アプリケーション・プログラミング・インタフェース (API) の 3 つの重要なアーキテクチャ機能を統合します。このフレームワークは、これらの重要な機能がアジリティを促進し、組織内の新しいプロセスを強化して競争上の優位性を生み出す方法を規定するものです。

旅行やホスピタリティなどの業界は、新たなビジネス手法によって変革を果たしました。現在は新しいサービスが提供されており、消費者がサービスを利用する方法も以前とは変わりました。この破壊的変化の傾向は、金融サービスから政府機関に至るまで、他の主要産業にも広がっており、ビジネスと顧客の相互作用に関する新たなテクノロジーと考え方によって高まっています。こうした新しいサービスを提供するために、従来型の組織は自社の IT テクノロジーを根本的に変革することを迫られています。

その流れについていくためには、ソフトウェアシステムの変更を迅速に計画および実行する能力が必要です。

現在求められるスピードでのソフトウェア提供には、アジャイルなインフラストラクチャ基盤が必要になります。ここでの「アジャイル」はアジャイルソフトウェア開発を指すのではなく、アジャイルの従来の意味、つまり、柔軟で、より迅速に動作できることを指します。

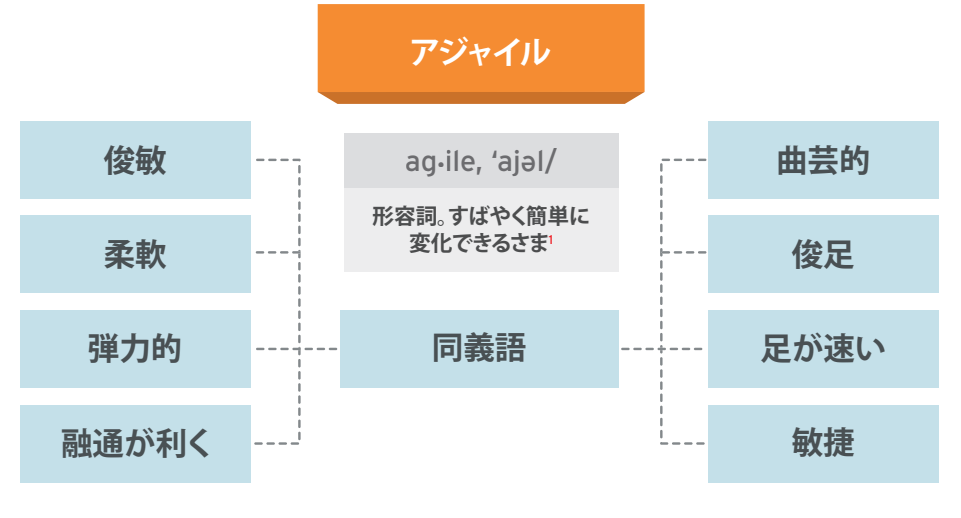

図 1. アジャイルの定義

1 オックスフォード英語辞典

「顧客を継続的に獲得してサービスを提供し、維持するためには、エンゲージメントシステムとレコードシステムの間のインタフェースがよりアジャイルになることが必要です。ここで言うアジャイルとは、スケーラビリティや、既存の API に新しい属性を追加したり、今後のためにより多くのコンテキストを提供したりするような、迅速に適応できる能力という意味でのアジャイルです。」

THE FORRESTER GROUP
HENRY PEYRET 氏

Henry Peyret 「TechRadarTM: Integration Technologies, Q2 2015」Forrester Research, Inc. 2015 年 6 月 23 日

アジャイル手法は今日までソフトウェア開発手法として注目され、アプリケーション作成方法の改善および最適化に活用されてきました。DevOps[2] 手法は、それをアプリケーションのデプロイに取り入れようとしました。

しかしこれまでのところ、DevOps そのものは多くの場合、主に組織が自社開発した新しいソフトウェア・アプリケーションに適用されるのみにとどまっています。

インフラストラクチャのアジリティはそれよりさらに大きなものであり、レガシーソフトウェアを含むすべての IT システムを包含する環境を作り出します。アジャイルなインフラストラクチャは、既存システムの複雑さ、さまざまなデータタイプ、データストリーム、および顧客の期待を取り入れ、それらを統合する方法を見つけるアプローチです。これは、本質的にはインテグレーションの問題です。

価格設定を一晩で変更したり、新しい製品を提供したりすることができる組織は、何段階にもわたる手動の検証手順を経て 3 カ月後にやっと公開に至る企業に比べて、圧倒的に有利です。

これが、アジャイル・インテグレーションです。インテグレーションは、インフラストラクチャのサブセットではありません。ハードウェアとプラットフォームを備えたデータとアプリケーションを含むインフラストラクチャへの概念的なアプローチです。インテグレーション・テクノロジーをアジャイルおよび DevOps テクノロジーと連携させることにより、チームが市場の要求に応じて迅速に変更できるプラットフォームの作成が可能になります。

計画至上時代の終焉：組織とアジリティ

「我々が知っている形での計画は、今や意味をなさなくなっています」とは、2017 Red Hat Summit で、Red Hat の CEO、Jim Whitehurst (ジム・ホワイトハースト) が行った基調講演でのメッセージです。「よくわからない環境での計画は効果的ではありません。」[3] ビジネス環境が迅速化し、変化が増えてくると、計画はすぐに破綻してしまうので、特定のアクションに縛られていると大きな代償が降りかかってきます。

つまり、情報が少ないほど、あるいは環境の安定性が低いほど、計画の価値は低くなります。

何を知らないのかがわからない

インフラストラクチャの計画には通常、長期的なアプローチが必要であり、時には数年にわたることもあります。数年にわたる計画では、市場の変化に合わせて革新したり、方向転換したりする能力が損なわれる可能性があります。Jim Whitehurst が示唆した「計画の無意味さ」は、より迅速な計画立案とそれらを実行する能力に帰着します。計画のスパンが短く、新しい計画を開拓する環境です。

6 カ月あるいは 24 カ月といったの長期の開発サイクルを習慣としているチームは、このような急激な変化に対応しきれない可能性があります。旧態依然とした構造の組織が、まったく新しい方法で市場にアプローチしている新興企業と競わなければならない場合、この問題はさらに大きなものとなります。Netflix と Blockbuster、または Uber と従来のタクシーサービスといったわかりやすい事例もありますが、スタートアップによる破壊的影響は、1993 年の Amazon や 1980 年代のパーソナルコンピューターに始まる情報化時代の最も初期の時期まで遡ります。

2 DevOps でイノベーションを加速 https://www.redhat.com/ja/insights/devops

3 2017 Red Hat Summit での Jim Whitehurst の基調講演 https://www.cbronline.com/news/enterprise-it/software/red-hat-ceo-planning-know-dead/

表 1: 各業種の破壊的革新

業種	従来のサービス	破壊的革新企業	影響
運輸	タクシー、公共交通機関	Uber、Lyft	小規模な地域密着型企業には再現することがほぼ不可能な均一化した顧客体験を創出
ウェルスマネジメント	投資会社	自動化されたファンド	ファンド管理の差別化要因を人員からアルゴリズムにシフト
小売	実店舗での買い物	Amazon	購買習慣をオフライン購入からオンライン購入に変更
検索エンジン	Google、ブラウザベースの検索	音声検索	Google の主要な販売チャネルに影響、新規参入の余地を作った

スタートアップと破壊的革新が持つ強みは、インフラストラクチャ、チーム、アプリケーション、アーキテクチャ、さらにはそれらのデプロイメントのプロセスを自由に構築できることです。革新的なアイデアを持っているというだけではなく、彼らはレガシー・インフラストラクチャ、あるいは Rachel Laycock 氏が冗談めかして言うところの「レガシーな人たち」[4] による制限を受けないので、そのアイデアを実行することができます。彼らはアジャイルです。

このような組織は、何か新しいものを構築する能力に加えて、変化に対応できるシステムも構築します。そうしたソフトウェア・インフラストラクチャは差別化に不可欠な要素であり、市場のニーズの変化に応じて、システムのほぼあらゆる部分を交換、更新、あるいは削除できます。新興企業でも時とともに適応能力が低下してくる場合がありますが、トップを争う組織はあらゆる方策を駆使して、変化する力を守ります。

課題に立ち向かう

急速に変化する環境で成功するためには、IT インフラストラクチャ全体が俊敏に機能しなければなりません。

変化は次の 2 つのレベルで生じる必要があります。

- アーキテクチャ設計からチームコミュニケーションに至るまでのアジャイルプロセスの組織的および文化的サポート
- 機能を迅速にアップグレード、追加、および削除する能力を生み出すテクニカル・インフラストラクチャ

技術的および文化的変化は、アジリティを生み出すものではなく、アジリティの基盤になるものです。

eBay のプロダクトマネージャー、Marty Cagan 氏は、すべてのプロジェクトに対して彼が税と呼ぶものを適用しています。新しいインフラストラクチャ・プロジェクトに取り組むために、あらゆるルーチンプロジェクトとは別の時間とリソースを確保し[5]、新しいプロジェクトとイノベーションを最優先事項とします。

4 Rachel Laycock「Continuous Delivery」Red Hat Summit – DevNation 2016 午後のゼネラルセッション 2016 年 7 月 1 日 カリフォルニア州サンフランシスコ https://youtube.com/watch?v=y87SUSOfgTY

5 Marty Cagan「Inspired: How to Create Products Customers Love」Wiley Press 2017 年

「市場投入までの時間とアジリティで競合を打ち負かすことができなければ、負けは決定的です。新しい機能は常にギャンブルです。運が良ければ、リリースしたうちの 10% で利益を得ることができます。ですから、これらの機能の市場投入とテストの実施が早いほど、勝ち目は大きくなります。ついでに元金回収も迅速になります。つまり、ビジネスの収益化が早くなるということです。」

THE PHOENIX PROJECT
GENE KIM 氏

Gene Kim、Kevin Behr、George Spafford「The Phoenix Project: A Novel about IT, DevOps, and Helping Your Business Win」オレゴン州ポートランド：IT Revolution Press 2013 年

アジリティのインフラストラクチャ

さまざまなグループがバラバラな方向性で改善方法を模索する状況では、新しいテクノロジーを次々に使用しても、アジャイルなインフラストラクチャの作成にはあまり役立ちません。首尾一貫したトップレベルの目標がなければ、どの新機能の組み合わせが組織の全体的な機能に真の違いをもたらすのかを識別することは困難です。

アジャイル・インテグレーションの 3 つの柱

アジャイル・インテグレーションのアプローチは、3 つの主要なテクノロジーによって支えられています。

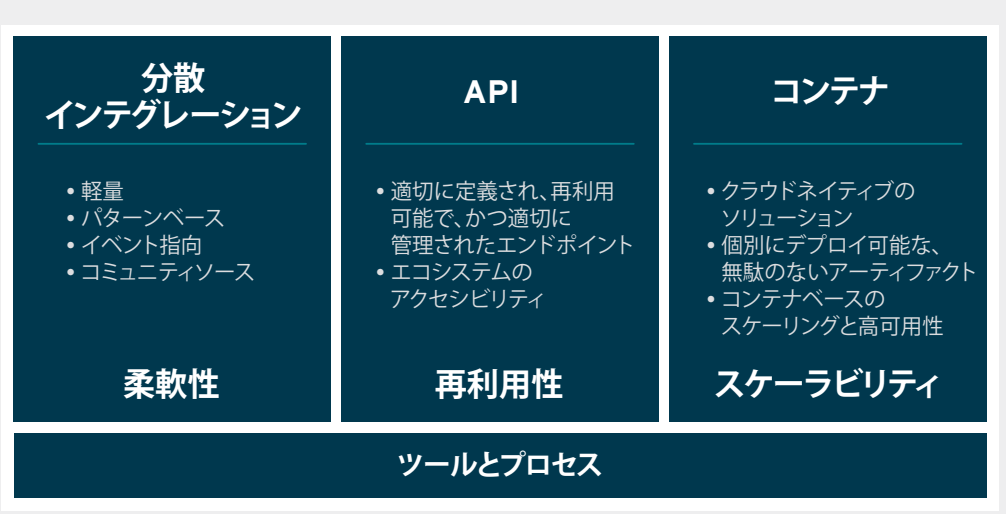

図 2. アジャイル・インテグレーションの 3 つの柱

1. **分散インテグレーション：** エンタープライズの作業とデータフローを反映した数十のハイレベルな統合パターンを用意します。これらの統合パターンをコンテナ内にデプロイすると、特定のアプリケーションやチームに必要なスケールで必要な場所に統合パターンをデプロイできます。これは従来の集中型インテグレーション・アーキテクチャではなく、分散インテグレーション・アーキテクチャであり、各チームが必要な統合パターンをアジャイルに定義およびデプロイすることを可能にします。
2. **API：** 安定し、適切に管理された API は、チーム間のコラボレーション、開発、運用に多大な効果をもたらします。API は、安定した再利用可能なインタフェースで主要なアセットをラップし、そのインタフェースが組織全体で、またはパートナーや顧客とともに再利用するためのビルディングブロックとして機能するようにします。API はコンテナとともに多様な環境にデプロイできるため、さまざまなユーザーがさまざまな API セットと対話することができます。
3. **コンテナ：** API と分散インテグレーション・テクノロジーの両方で、コンテナは基盤となるデプロイメント・プラットフォームとして機能します。コンテナを使用すると、開発、テスト、および保守が容易かつ一貫した方法で、特定の環境内に正確なサービスをデプロイできます。コンテナは DevOps 環境とマイクロサービスの主要なプラットフォームなので、コンテナをインテグレーション・プラットフォームとして使用すると、開発チームとインフラストラクチャチームの関係は透明性が高く協力的なものになります。

これらの 3 つのテクノロジーは、IT インフラストラクチャのアジリティ向上を実現するものです。というのも、これらの機能がそれぞれ抽象度を上げることで、さまざまなチームの協業が可能となるためです。API と分散インテグレーションを備えたコンテナプラットフォームを使用すると、インテグレーションそのものからインテグレーションの実装が抽象化されます。API と分散インテグレーション・パターンにより、多くの人に理解できるレベルで特定のアセットがパッケージ化されるため、チームのアジリティが向上します。基盤となるインフラストラクチャを理解または変更する必要はありません。

これらの各テクノロジーはそれぞれ、特定のインテグレーションの課題に大きなアジリティをもたらします。併用すると、相乗効果が得られます。テクノロジーを強調することは文化です。DevOps のプラクティス、特に自動化やデプロイメントのプロセスと組み合わせることで、テクノロジーのメリットが高まります。

分散インテグレーション

現在の IT システムの最大の課題の 1 つは、組織全体に散らばったアプリケーションを接続する必要があることです。インテグレーション・サービスの難しさは、ますます複雑で集中化したインテグレーションハブを生み出しています。多くの場合、エンタープライズ・サービス・バス (ESB) として実装されているこれらのハブは、急速な変化に対応できる柔軟性に乏しく、非常に複雑なボトルネックになっています。

分散インテグレーションは、前世代の ESB と同じ技術目標の多くを達成できますが、その方法は組織内のチームに対してより適応するものです。ESB と同様、分散インテグレーション・テクノロジーは、変換、ルーティング、解析、エラー処理、およびアラート機能を提供します。違いは、インテグレーションのアーキテクチャです。

分散インテグレーション・アーキテクチャは、各インテグレーションのポイントを、より大きな一元化された統合アプリケーションの一部ではなく、個別の一意のデプロイメントとして扱います。組織全体にデプロイされた他のインテグレーションに影響することなく、インテグレーションを特定のプロジェクトまたはチーム向けにローカルにコンテナ化してデプロイすることができます。この分散アプローチにより、アジャイルプロジェクトに必要な柔軟性が得られます。また、基盤となるコンテナプラットフォームを使用することにより、アジャイルチームまたは DevOps チームと同じツールチェーンを使用し、チームが自身のツールやスケジュールとの統合を管理する能力を高めます。これは基本的にインテグレーションをマイクロサービスとして扱い[6]、開発とリリースのインテグレーションを迅速化します。

これには、開発者ツールおよびプロセスとの連携が不可欠です。従来のソフトウェア・インストラクチャは 1 つの部門の特殊なユーザーセットによって開発および管理され、ソフトウェア開発プロセスとは別に展開される集中型のものでした。しかし分散インテグレーションはそれとはまったく異なるアプローチであり、それが核となっています。共通のプラットフォームとツールを使用してインテグレーション・アーキテクチャを配布することにより、プロジェクトレベルですべての開発者がアクセスできるようになり、インテグレーションが必要な時にはいつでもどこでも軽量のデプロイメントを実行できます。

「ソフトウェア分野では、何か手間のかかる作業をしなければならない時、その手間を減らすために有効なのは、その作業の頻度を減らすことではなく、増やすことです。」

「CONTINUOUS DELIVERY: RELIABLE SOFTWARE RELEASES THROUGH BUILD, TEST, AND DEPLOYMENT AUTOMATION」
DAVID FARLEY 氏

David Farley、Jez Humble「Continuous Delivery: Reliable Software Releases Through Build, Test, and Deployment Automation」Addison-Wesley Professional 2010 年

表 2. ソフトウェア・ライフサイクルの各段階におけるインテグレーション・テクノロジーの比較

ライフサイクルの段階	ESB、ほとんどのIPAAS (INTEGRATION PLATFORM-AS-A-SERVICE)	分散インテグレーション・テクノロジーのサポート
バージョン管理	プロプライエタリー	GitHub、その他
ビルド	プロプライエタリー	Maven、その他
デプロイ	プロプライエタリー	コンテナとその他の DevOps ツール
管理とスケーリング	プロプライエタリー	コンテナとその他の DevOps ツール

ESB を使用するためには、開発環境と運用環境で使用されているあらゆるツールに加えて、ライフサイクル全体でその ESB のツールを使用する必要があります。この制限により、運用は扱いにくく非効率的でエラーが発生しやすいものになってしまいます。

メッセージングによるインテグレーションの強化

アーキテクチャ上、分散インテグレーションは、インテグレーションをマイクロサービスとして扱います。マイクロサービスはコンテナ化が可能で、簡単かつローカルにデプロイでき、迅速なサイクルでのリリースが可能です。

インテグレーション・テクノロジーは、この種の軽量なマイクロサービスベースのアーキテクチャをサポートできなければなりません。Red Hat® Fuse によって、ユーザーはインテグレーションをコードとして扱うことができ、コンテナ内を含めてどこででも実行することができます。

さらに、Fuse は Red Hat JBoss AMQ にバンドルされており、メッセージング・インフラストラクチャを提供します。強力なメッセージング・インフラストラクチャにより、イベントとデータがシステム間で効果的にルーティングされます。メッセージングはマイクロサービスを備えた重要なアーキテクチャツールです。なぜなら、メッセージングの非同期の性質には依存関係がないからです。

このインテグレーションとメッセージングの組み合わせは、より効果的なルーティング、複数の言語とプロトコルのサポート、非同期スループット、およびデータ管理の改善によって、インテグレーション・アーキテクチャの全体的なパフォーマンスを向上します。

「しばしばデジタル変革と呼ばれる新たな競争相手が、組織によるITアーキテクチャの再考、オンプレミスのインフラストラクチャ、クラウドなどへのワークロードの再分配、および相互運用によるビジネス戦略と運用のサポートの必要性を高めています。このような変化すべてがインテグレーションへの新しいアプローチを必要としており、このアプローチを『ハイブリッド・インテグレーション』と呼んでいます。」

451 GROUP
CARL LEHMANN 氏

Carl Lehman, 451 Research「The Disruptive Role of Integration PaaS and APIs in the New Hybrid Integration Platform Market」2017年7月 https://451research.com/report-long?icid=3862

トレンドについていく

コンテナの導入は拡大していますが、どの程度、そしてなぜ拡大しているのでしょうか。451 Researchは、市場で250%の成長を予測していますが[7]、その数値は支出を表すもので、デプロイメントの拡大規模ではありません。実際のデプロイメントを測定するのは少々困難です。Red Hatが委託したBainの調査では、現在顧客の約20%が実稼働環境でコンテナをデプロイしており、開発環境とテスト環境でほぼ同じ数値を示すことがわかっています。しかし、コンテナの評価や概念実証を行っている割合は30%を超えています。[8]

このように不明瞭な結果になる理由の一端は、「コンテナを使用する」という表現の指す内容のあいまいさにあります。Enterprisers Projectでは、4つの異なるコンテナ導入のパターンを概説しています。そのパターンとは、一般的な開発あるいはデプロイメントのプラットフォームとして使用、クラウドネイティブまたはマイクロサービスのプラットフォームとして使用、ハイブリッドクラウド内での使用、イノベーションプロジェクトでの使用の4つです。[9] コンテナの使用方法により、「導入済みである」と考えるかどうかが変わってしまう可能性があるのです。

アジャイル・インテグレーションのアイデアは、従来の運用をサポートできるインフラストラクチャ・プラットフォームを作成することです。このプラットフォームは、すべての実装パターンから借用することができますが、中核的な機能は、プラットフォーム、すなわち新しいプロジェクトと既存のサービスの両方の基盤となることであり、これは変わりません。

コンテナ

仮想化、クラウド、およびコンテナは、同様の目標を持つ類似のテクノロジーです。これらのテクノロジーは、物理ハードウェアからソフトウェアのオペレーティング環境を抽象化するため、ハードウェア上でより多くのインスタンスをスタックし、使用率、スケール、デプロイメントをより効率的に管理できます。しかし、それらが課題に対処する方法は異なります。仮想化は、オペレーティングシステム層を抽象化します。クラウドは、永続的な専用サーバーインスタンスの概念を不要にします。コンテナは、単一のアプリケーションを実行するのに十分なオペレーティング環境とライブラリを定義します。

コンテナ技術を活用した、より規範的で軽量なアプローチにより、コンテナは最新のソフトウェア環境にとって理想的なツールになっています。各インスタンスは、オペレーティングシステムから各ライブラリに含まれる正確なバージョンまで、不変の定義を使用します。インスタンスごとに環境の再現性と一貫性が高くなるため、継続的インテグレーションと継続的デリバリー(CI/CD)のパイプラインに最適です。さらに、コンテナイメージは単一のアプリケーションに必要なもののみを定義するため、コンテナはマイクロサービスに一致し、コンテナ・オーケストレーションは大規模なマイクロサービス・インフラストラクチャのデプロイメントと管理も調整することができます。

軽量で再現性のある組み合わせにより、コンテナはアジャイル・インテグレーションに最適なテクノロジー・プラットフォームになります。

7 Cloud-Enabling Technologies Monitor レポートに基づく451 Research インフォグラフィック 2017年1月 https://451research.com/images/Marketing/press_releases/Application-container-market-will-reach-2-7bn-in-2020_final_graphic.pdf

8 「Bain survey: For Traditional Enterprises, the Path to Digital and the Role of Containers」2016年11月 https://www.redhat.com/ja/resources/path-digital-containers

9 https://enterprisersproject.com/article/2017/8/4-container-adoption-patterns-what-you-need-know

従来のインテグレーション・アプローチは、高度に集中化した構造であり、ESB はインフラストラクチャの主要ポイントに配置されていました。分散インテグレーションと API 管理の両方に、特定の場所またはチームに必要な機能のみをデプロイする分散型アーキテクチャがあります。コンテナは、不変の性質により環境間でイメージとデプロイメントの一貫性を維持し、両方のアプローチの基盤となるプラットフォームとして機能するので、不透明な依存関係や不一致がなく迅速にデプロイまたは置換できます。

分散アーキテクチャの鍵は、それがインテグレーションと API のどちらを備えたものかに関わらず、複雑な承認プロセスなしで新しいサービスを設計およびデプロイする方法が必要だということです。

コンテナによって、分散インテグレーションと API をマイクロサービスとして扱うことができます。開発チームと運用チームの両方に共通のツールを提供し、管理されたリリースプロセスで迅速な開発プロセスを使用する機能を提供します。

コンテナにはオーケストレーションが必要

マイクロサービスが単一の個別の機能に相当するように、各コンテナは単一のサービスあるいはアプリケーションに相当します。マイクロサービス・アーキテクチャでは、数十または数百もの個別のサービスが存在する可能性があり、それらは、開発環境、テスト環境、および本番環境にわたって複製されます。

それほど多数のインスタンスを扱う場合、コンテナ環境が効果的に機能するためには、インスタンスをオーケストレーションして高度な管理タスクを実行する機能が欠かせません。

Red Hat OpenShift は、Docker コンテナを Google の Kubernetes オーケストレーション・プロジェクトと組み合わせます。また、インスタンス管理、監視、ロギング、トラフィック管理、自動化などの集中管理機能も提供しますが、このような機能は、コンテナだけの環境ではほぼ実現不可能です。

Red Hat OpenShift は、セルフサービスカタログ、インスタンス・クラスタリング、アプリケーションの永続性、プロジェクトレベルの分離など、開発者にとって使いやすいツールも提供します。

この組み合わせにより、特に安定性やテストといった運用の要件と開発者ニーズのバランスが取れ、使いやすさと迅速なデリバリーが可能になります。

API

ほとんどの情報インフラストラクチャは、数百あるいは数千ものシステム、アプリケーション、アセットを包含していますが、これらのシステムが相互作用することは非常に困難になる可能性があり、IT 管理者がどのシステムが利用可能かさえわからないこともあります。

API は、インテグレーション・テクノロジーを使用して接続できるすべてのアセットのインタフェースであり、アプリケーションが相互に通信する方法を設定する一連の定義またはルールです。

組織が、集中型のインテグレーション・テクノロジーを中心としたアプローチから分散型アプローチに移行するにつれて、セルフサービスが重要な優先事項になります。アジャイルチームには、社内外で開発されたサービスを探し、テストし、使用するための権限と自律性が必要です。強力な API 機能が、この権限と自律性をチームに与えます。API によって、チームは必要なインテグレーションを実現し、組織はセキュリティ、承認、使用ポリシーを確実に管理および実行できます。また、API はチームに対し、インテグレーションの設計方法に関するレファレンスを提供します。

API は最終的なアプリケーションとは異なります。アプリケーションがどのように相互作用するかを定義するものであり、個々の開発者は API をプロジェクト内でビルディングブロックとして使用します。API は、開発者とチームの共通言語となります。組織は、サービスの新しく革新的な使用を生み出すために共有および協力するコミュニティを育成することにさえも、API を使用できます。

利用者層ごとにさまざまな API や API のサブセットを提供することもできます。ベンダーのニーズは、内部の開発チームやコミュニティ開発者のニーズとは必ずしも一致しません。API 管理には、アプリケーションやユーザーグループ用の API の設計、および API のライフサイクルの管理が含まれます。API は製品として管理されることが多くなり、API ごとに異なるチームが責任を負いますが、これらすべてのリソースで統一性と使いやすさを確保する必要があります。

分散インテグレーションと同様、コンテナは、API 開発をより大きな開発および運用プロセスやツールと連携する方法で、API を開発、デプロイ、管理するためのプラットフォームとして機能することができます。

適切な API プラットフォームは開発者の生産性を高める

API が大きなパワーを持つのは、社内外を問わず多数のユーザーがその API を使用できるからです。Red Hat 3scale API Management Platform はそうしたすべてのユーザーを支援するツールであり、開発者が API の作成に共同で取り組むための開発者ポータルとそれらの API を公開できる管理者ポータルを提供します。

3scale API Management Platform は、認証提供、主要なクラウドプロバイダーとのインテグレーション、コンテナ内での実行により、これらの API を外部ユーザーが使用できるようにします。

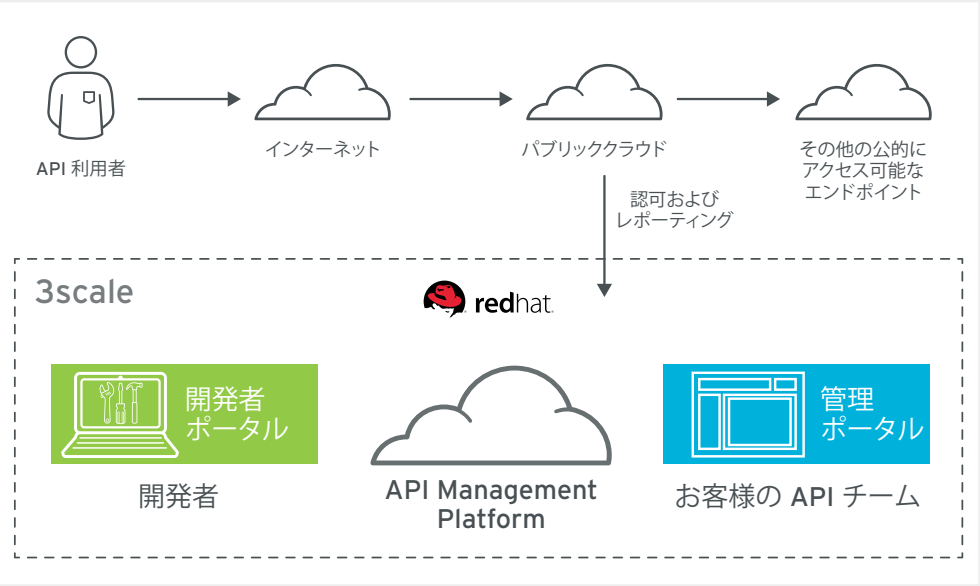

図 3. API 管理、エンドポイント、パブリッククラウドの概念図

API 戦略とは、API 設計とその API を公開する方法を組み合わせたものです。3scale API Management Platform、特にコンテナプラットフォーム上の 3scale は、その戦略を実行する手段となります。

アジャイル・インテグレーションのアーキテクチャ

チーム体制

アジャイル・インテグレーションの柱となるテクノロジーは、デプロイされ、再利用可能な機能としてチームが利用できるようになったときに最大限の効果を発揮します。

ここでいう機能とは、承認されたグループがテクノロジーをセルフサービスで使用し、組織のガイドラインに容易に従い、ベストプラクティス情報にアクセスすることができる機能のことを指します。情報アーキテクトや IT 管理者は、個々のチームに対して次のような明確なプロセスを定義する必要があります。

- 広く利用可能な使用ガイドラインを提供する。
- 必要に応じて使用およびベストプラクティスのルールを適用しながらも、それらのルールを超えた自由な実験を許可する。
- プロトタイプからテスト、実稼働、更新、廃止までの移行プロセスを明確に定義する。
- 新しいデプロイメントと開発のための情報共有を許可する。
- インフラストラクチャチームを全プロセスにかかわらせるのではなく、セルフサービス機能のイネーブラーおよびプロバイダーとして使用する。

たとえば、ソフトウェアチームに対しては、新しい API を開発、テスト、準備して完全なセルフサービス方式で公開できるようにする必要がありますし、他のグループへの通知やドキュメンテーション更新のプロセスも必要です。公開や実稼働に移行する前に、何らかの処理や他のチームとのクロスチェックが行われる場合がありますが、このインフラストラクチャではそのプロセスを可能な限り自動化すべきです。

インフラストラクチャ・アーキテクチャ

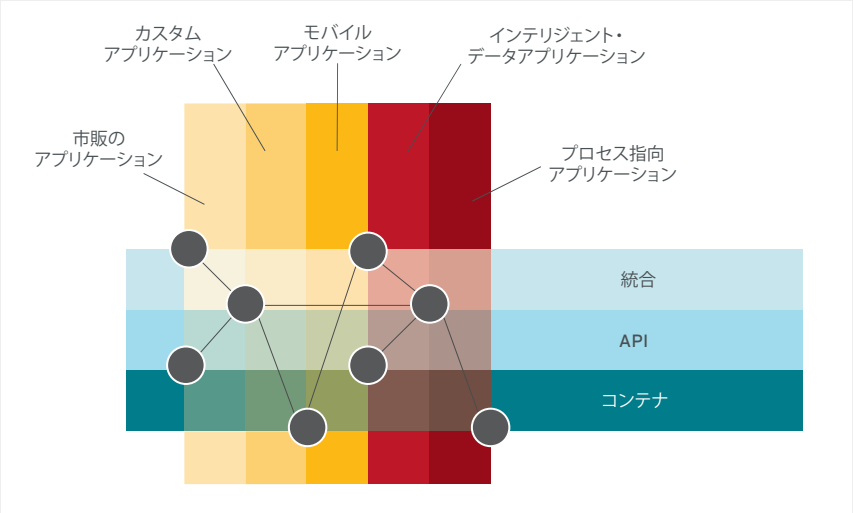

図 4. アジャイル・インテグレーション・テクノロジーとアプリケーションスタックの相関図

コンテナ、API、インテグレーションは連携して、組織の内部ソフトウェアエコシステムの強固な基盤層となります。その基盤層とは多くの場合、外部インテグレーションのアクセスポイントです。

異なるタイプのシステムがさまざまな再利用可能なエンドポイントを公開します。各エンドポイントは、再利用可能な API として表示され、その多くがコンテナ内で実行され、スケーラビリティと容易なデプロイメントを実現します。インテグレーションは、個々のサービスのグループを統合するか、組織のさまざまな部分から結果を収集することにより、システム全体で必要な場所に変換、構成、あるいはインラインのビジネスロジックを提供します。

統合アプリケーションは、エンドユーザー・アプリケーションを供給する前にさらに集約することができます。

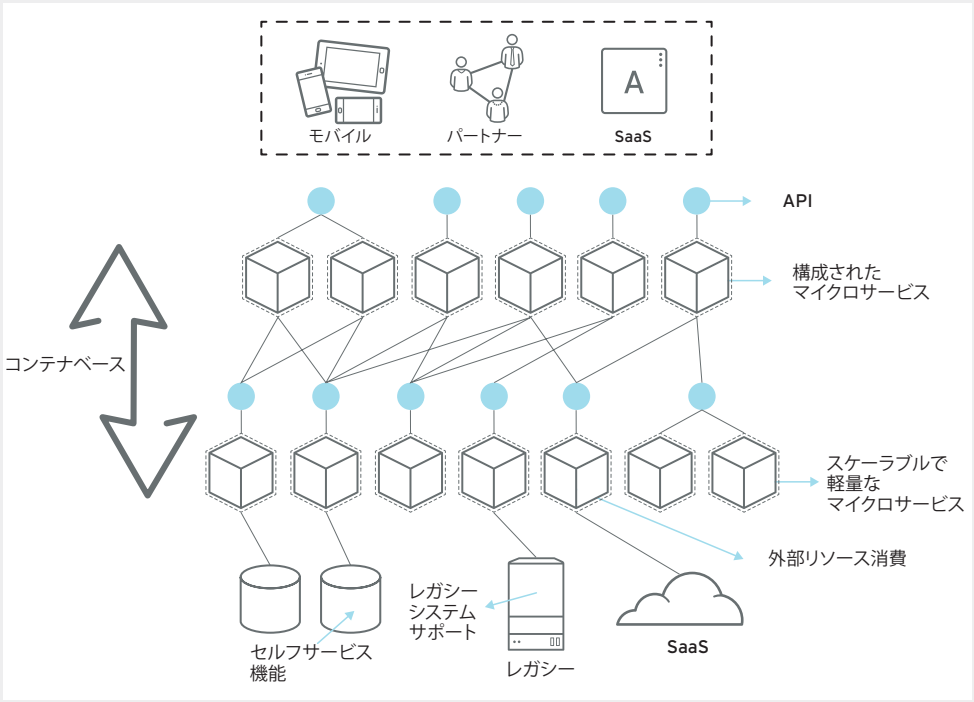

図 5. コンテナ、API、および分散インテグレーションによるインフラストラクチャ設計

すべてのシステムを小さな断片に分割したり、API 抽象化の複数のレイヤーを通過させたりすることは想定していません。そのような運用は、効率の低下、遅延の増加、不要な複雑さの追加につながる可能性があります。一部の分野では、既存のレガシー ESB 機能を保持して特定のアプリケーション間の接続を保持することが適切な選択になる場合があります。分散システム間の依存関係も、適切なツールを使用して追跡および管理する必要があります。

ただし、システム全体では、コンテナ、API、およびインテグレーションの観点からアーキテクチャを作り直すことが、各サービス、統合ポイント、顧客の対話にとって正しい選択になり得ます。たとえば、大量のインバウンドリクエストを処理する場合、ボトルネックとなりうる単一の ESB を経由させることなくセキュリティをチェックし、適切なバックエンドサービスに直接ルーティングすることもできます。

また、ハイブリッドの分散クラウド環境では、問題とされるバックエンドシステムの多くが、物理的に異なる場所に存在する場合があります。そのような状況では、主要なビジネスロジックを保持する単一のセントラル・インテグレーション・システムを介してすべてをルーティングするよりも、地理的に近くにあるシステムを統合してローカルのニーズに対応するほうが、効率的にもセキュリティ的にも有効です。

アジャイルな組織と文化

インフラストラクチャのライフサイクルは、ソフトウェア開発や運用のライフサイクルとは大きく異なります。開発のサイクルとは、1 つのプロジェクトを完了してから次のプロジェクトに取りかかることであり、効率とは、製品のリリースを加速し、与えられた時間に生産できる機能の数を増やすことを意味します。メンテナンスと安定性を重視した運用の場合でも、セキュリティパッチと更新の適用や新しいサービスのデプロイメント、あるいは変更のロールバックをより効率的かつ迅速に行うことが依然として有益です。

しかし、インフラストラクチャのアプローチはまったく異なるものです。インフラストラクチャは、特定のソフトウェア・エンジニアリング・プロジェクトに取り組む機能横断型チームとは大きく違う、非常に専門性の高い異なるグループが、より長い時間枠で作業する傾向があります。インフラストラクチャ・プロジェクトは通常、ソフトウェアプロジェクトよりもはるかに規模が大きいため、短いリリースサイクルでは十分に遂行できなかったり、不完全なままになってしまう可能性があります。エンタープライズ IT プロフェッショナルの Andrew Froelich 氏が InformationWeek で述べたように、インフラストラクチャには、特にハードウェアとデータセンターの場合、それ以上進めると元に戻せなくなるポイントが存在します。パブリッククラウドの場合でも同様に、そのラインを超えるとプロジェクトを破棄してやり直すことができなくなるポイントがあります。[10] 一度構築したインフラストラクチャはずっと残ります。しかし、インフラストラクチャのパフォーマンスにあわせて手法を調整することは可能です。

アジャイルや DevOps のようなレスポンシブで反復的なプロセスの利点は、開発チームと運用チームにとっては明らかですが、インフラストラクチャチームにとってはそれほど顕著ではありません。Froehlich 氏はインフラストラクチャにアジャイル手法を適用することの利点と欠点を分析しましたが、この分析は 1 つの重要な側面を見逃しています。それは、アジャイル手法を用いると、インフラストラクチャチームは開発チームおよび運用チームと連携できるようになるということです。Rohan Pearce 氏は CIO 誌で、インフラストラクチャチームを機能チームではなくアジャイルスタイルのワークセルに変更することについて述べています。[11] Telstra Enterprise Services のチームは、システムをチェックアウトするプロセスや更新を行うプロセスが非常に困難で複雑だったため、開発者グループには内部システムをただ無視させていましたが、作業グループを調整したことにより、サイクルタイムが 212 日から 42 日に短縮されました。[12]

この例は、ポイントを押さえたプロセス変更を行うことで、インフラストラクチャチームが内部グループに対して、どれだけ効果的にサービスを提供できるようになるかを示しています。

アジャイル・インテグレーション・テクノロジーは、よりアジャイルなインフラストラクチャを支えるものです。API、コンテナイメージ、分散インテグレーションは、ソフトウェア・インフラストラクチャについて語る際の新たな言語になります。

10 Andrew Froehlich「Should IT go agile? The pros and cons」2015 年 10 月 6 日
http://www.informationweek.com/infrastructure/pc-and-servers/should-it-go-agile-the-pros-and-cons/d/d-id/1322448

11 Ronan Pearce「Can infrastructure be agile?」2013 年 6 月 20 日
https://www.cio.com.au/article/465436/can_infrastructure_agile_/

12 http://agilemanifesto.org/

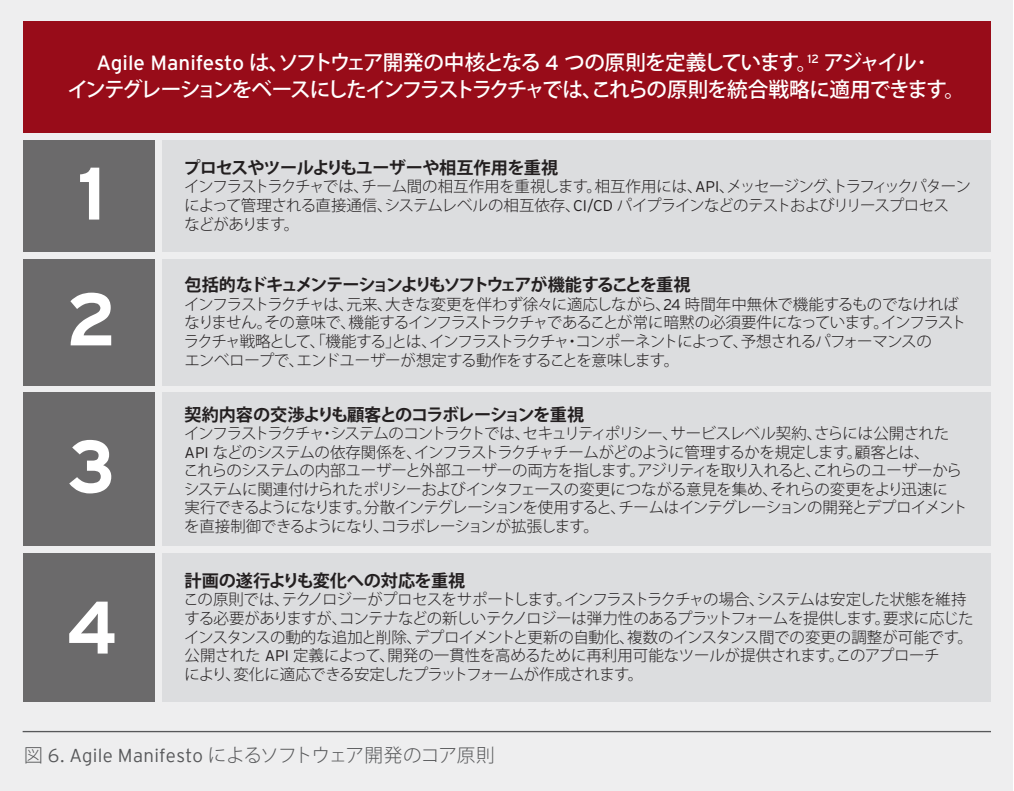

図 6. Agile Manifesto によるソフトウェア開発のコア原則

アジャイル・インテグレーションは、テクノロジーを使用してインフラストラクチャチーム内の文化的変化をサポートします。インフラストラクチャ戦略の基盤として機能し、インフラストラクチャ・テクノロジーとそのチームが、開発およびビジネス戦略とより密接に連携できるようにします。

アジャイル手法は、個人、ビルド、依存関係など、ソフトウェアプロジェクトの重要な部分を特定し、これらの要素間の関係を定義できます。アジャイルプロジェクトとしてインテグレーション・インフラストラクチャにアプローチする場合、チーム、コンテナイメージ、API、統合ポイントなど、アジャイルによって定義されたものと並行して識別できる類似の要素と関係性があります。表 3 に、これらの類似点の一部を示します。

表 3. ソフトウェアアジャイルとインフラストラクチャ・アジャイルの要素の比較

プロジェクト	組織	詳細
個人	チーム	チームはそれぞれ、インフラストラクチャの特定の部分を担当します。これにより、チームが管理するシステムやAPI、チームリーダー、チームの目標など、チームの責務に関する情報が特定されます。
モジュール	API	明確に定義されたインタフェース (API) は、長期にわたって安定しており、独自のロードマップを持ち、特定のチームによって実行され、組織内で重要な特定の機能を作成します。
ビルド	コンテナイメージ	リリースは、テスト済みでタグが付けられた、デプロイ可能なユニットをベースとしており、アクセスできる任意のチームによって確実にデプロイすることができます。これは、モノリシックなバージョン管理されたコードを置き換えます。
依存関係のコンパイル	インテグレーション	これらの分散システムにおける異なるコンポーネント間のインテグレーションとマッピングを識別する要素です。この統合ポイントは、システムのあらゆる他の部分と同様に管理、作動、廃止、バージョン管理、およびテストすることができます。
ビルドのテスト	インフラストラクチャの自動化	ソフトウェアビルド、パフォーマンス、ユーザー要件をテストする機能から、複数のシステムの運用および監視までの完全なライフサイクル管理です。

インフラストラクチャの計画にアジャイルの原則を適用する

ほとんどの変更管理アプローチでは、すべてのサブシステムの包括的なドキュメンテーションが必要です。このドキュメンテーションは、監視方法からパフォーマンスパラメータ、担当チームまで、システムのあらゆる側面を詳細に網羅する必要があります。アジャイルの原則にはコラボレーションと適応性が必要であり、これはドキュメンテーション中心の変更管理とは矛盾します。

したがって、アジャイル手法を適用する場合は潜在的なすべての利害関係者、変更、システムコンポーネントを規範的に定義するのではなく、変更要求と計画の評価に使用できる一連のガイドラインと基準を定義します。次の質問について考えてみてください。

- どのようなエンドツーエンドのエクスペリエンスをユーザーに提供するのか。
- 関係する要素 (各チーム、API、システム) は、このエクスペリエンスの向上にどのように貢献しているか。また、その貢献は時間とともにどう変化していくか。
- サービスレベルを維持するために、監視とアラートをどのように、どのパラメーターに対して定義するか。
- 予想される動作を確認するために、どのような自動テストが必要か。
- チームがユーザーエクスペリエンスを中断せずに、新バージョンのサブシステムをテストおよびデプロイするためのリリースパイプラインは何か。
- コンポーネントサービスで障害が発生した場合、システム全体のサービスレベルにどのように影響するか。

アジャイル・インフラストラクチャ内の変更管理は、コントラクトではなく継続的なコラボレーションである必要があります。

プロジェクト成功の可能性

IT プロジェクトが成功する可能性はどのくらいでしょうか。それを知るにはまず、成功の判断基準を定める必要があります。それは、仕様を満たすこと、顧客の導入を増やすこと、それともとにかくリリースにこぎつけることでしょうか。プロジェクト管理のトレーニンググループ 4PM は、プロジェクトが予算内、期限内、仕様どおりに完了することを成功と定義しました。[13] そしてその定義に照らすと、IT プロジェクトの約 70% は失敗とみなされると算出しています。[13] しかしこの数値は変化し始めています。Project Management Institute が実施した最近の調査では、過去 5 年間と比較すると、より多くのプロジェクトが計画目標を達成していることが明らかになりました。[14] IT チームとビジネスチームの連携が強化されたことにより、戦略と顧客のニーズに関するより良い情報を得られるようになった結果だと考えられます。[8]

その戦略的連携の理由の 1 つが、アジャイルチームの実装です。アジャイルは、コラボレーションとフィードバック、問題とシステムの全体像、創造的なアプローチを奨励します。

13 4PM.com「Why projects fail so often」2015 年 9 月 27 日
http://4pm.com/2015/09/27/project-failure/

14 Sharon Florentine「IT project success rates finally improving」2017 年 2 月 27 日
https://www.cio.com/article/3174516/project-management/it-project-success-rates-finally-improving.html

技術スタックを共有すると、議論の中心は個々のコードではなくシステムとその相互依存性へと移ります。これはシステムレベルの考え方であり、社内開発のソフトウェア、ベンダーシステム、その間の接続を含むソフトウェア・インフラストラクチャのコレクション全体を単一のシステムとして扱います。API とメッセージングシステムは、インフラストラクチャ全体に広がり、ソフトウェアシステムを一元化するために機能します。

API と分散インテグレーションは、個々の開発や運用チーム内で開発および理解できるため、インテグレーションに対するチームの責務に関する知識ははるかに明確になります。インテグレーション自体は、システムとアプリケーションの間の相互依存関係が開発とデプロイメントを処理するチームによって認識されるため、よりよく理解されます。

インテグレーションをインフラストラクチャの基盤として使用し、そのインテグレーションに対する責務をチーム間で分散させることにより、アジャイルアプローチがより適切なインフラストラクチャ環境が作成されます。

まとめ：アジャイル・インテグレーションのデリバリー

アジリティはプロセスであり、プロジェクトではありません。

組織にとって、市場の変化に対応する能力はかつてないほど重要になっています。そして、新しいサービスを立ち上げたり、既存サービスを迅速に更新したりする機能は、主に IT システムが提供することになります。IT インフラストラクチャはデジタルサービスの基盤であるため、その再考はかつてなく大きな意味を持ちます。

インフラストラクチャチームは、リスクを軽減し、安定性を維持する必要があるため、今までは最適とは言い難い非常に長いプロセスに縛られてきました。しかし、インフラストラクチャの考え方をハードウェアまたはプラットフォームベースからインテグレーションベースにシフトすることは可能です。インテグレーションは、インフラストラクチャのサブセットではありません。ハードウェアとプラットフォームを備えたデータとアプリケーションを含むインフラストラクチャへの概念的なアプローチです。

Red Hat では、このアプローチをアジャイル・インテグレーションと呼んでいます。これは、インテグレーション・テクノロジーを使用して、よりアジャイルで適応性のあるインフラストラクチャを作成する方法です。アジャイル・インテグレーションには、3 つのテクノロジーの柱があります。

- **分散インテグレーション**は、メッセージングおよびエンタープライズ統合パターンを使用して、データとシステムを統合します。これらは、必要に応じてプロジェクトとタッチポイント全体に分散される小さなチーム主導のインテグレーションに分割されます。
- **内部 API 管理**は、再利用可能な一連のインタフェースを作成し、開発チームがアプリケーションやシステムと連携できるようにします。API は、アプリケーションがどのように対話すべきかについてのガイダンスと構造を提供します。
- **コンテナ**は、インテグレーション・プロジェクトを開発および運用プロジェクトと密接に連携させ、DevOps 手法を使用したソフトウェアプロジェクトと同様に、インテグレーションを開発、テスト、リリースできるようにします。

テクノロジーは、文化的変化を支える形で使用しなくてはなりません。そしてそれは、ソフトウェアだけでなく、インフラストラクチャチームのアジリティを高めることを意味します。インフラストラクチャチームがアジャイルの原則を取り入れていくに従い、これらの変更をサポートするテクノロジーを徐々に導入することができます。たった 1 つのプロジェクトで組織全体を再編成してアジャイルにすることなど決してできません。1 つのアジャイル・インテグレーション・テクノロジーを実装するか、ビジネスの 1 つの領域を変更してから、それらの変更を段階的に拡張する方が効果的です。

変更に対する IT インフラストラクチャの応答性の向上は、長期的な戦略目標です。前進していくのに、組織全体にわたる広範な変更を行う必要はありません。場合によっては、単独で変更を加えてからロールアウトする必要さえありません。

アジャイル・インテグレーションは、技術的および組織的なフレームワークの提供を通じて、IT インフラの変革を支援します。

RED HAT について

エンタープライズ・オープンソース・ソフトウェア・ソリューションのプロバイダーとして世界をリードする Red Hat は、コミュニティとの協業により高い信頼性と性能を備える Linux、ハイブリッドクラウド、コンテナ、および Kubernetes テクノロジーを提供しています。Red Hat は、新規および既存 IT アプリケーションの統合、クラウドネイティブ・アプリケーションの開発、Red Hat が提供する業界トップレベルのオペレーティングシステムへの標準化、複雑な環境の自動化、セキュリティ保護、運用管理を支援します。受賞歴のあるサポート、トレーニング、コンサルティングサービスを提供する Red Hat は、Fortune 500 企業に信頼されるアドバイザーです。クラウドプロバイダー、システムインテグレーター、アプリケーションベンダー、お客様、オープンソース・コミュニティの戦略的パートナーとして、Red Hat はデジタル化が進む将来に備える企業を支援します。

アジア太平洋
+65 6490 4200
apac@redhat.com

オーストラリア
1 800 733 428

インド
+91 22 3987 8888

インドネシア
001 803 440 224

日本
0120 266 086
03 5798 8510

韓国
080 708 0880

マレーシア
1 800 812 678

ニュージーランド
0800 450 503

シンガポール
800 448 1430

中国
800 810 2100

香港
800 901 222

台湾
0800 666 052

fb.com/RedHatJapan
twitter.com/RedHatJapan
linkedin.com/company/red-hat

jp.redhat.com
f11423_0518