e ブック

Red Hat

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

jp.redhat.com

目次

「IT のモダナイズはデジタル・トランスフォーメーションの中でも特に重要な部分で、イノベーションとパフォーマンスを迅速に向上させますが、同時に誤解が多い部分でもあります。トランスフォーメーションに欠かせない重要タスクとしては、その他にデジタルカルチャーの構築があります」[1]

Digital McKinsey
Michael Bender 氏および Paul Wilmot 氏

イノベーションにおけるインテグレーションの役割

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課題に対処できるよう進化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。

あらゆる業界がテクノロジーによって変革されました。現在では、ほとんどの企業が e コマースサービスを提供し、消費者はデジタルな手段で企業とやり取りすることが普通になりました。このような破壊的変化が進んだことで、組織は顧客が求める新しいデジタルサービスを競合他社よりもすばやく優れた品質で提供できるよう、IT 環境を根本的に変革するようになりました。

企業を超えてアプリケーションとサービスをつなげる新しい手段の基本となるソリューションが、アジャイル・インテグレーションです。アジャイル・インテグレーションは、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)、コンテナというアーキテクチャ上の 3 つの強力な機能を組み合わせ、アジリティを促進し、新しいプロセスを強化し、最終的に競争上の優位性を確保します。

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。

デジタルビジネスに欠かせないインテグレーション

卓越したカスタマーエクスペリエンスの提供は、単に差別化の問題ではなく、生き残りを賭けた課題になりました。個々のエクスペリエンスや対応が顧客ロイヤルティを築くブロックであるなら、全体的なカスタマージャーニーはこれらのブロックをつなぎ合わせるモルタルと言えます。

ホテルを例にとると、宿泊体験は、予約時の対応、宿泊客からの Wi-Fi の問題についての問い合わせへの対処、ロイヤルティプログラムなどと同じく重要です。デジタル経済においては顧客の要求が強まり、個別化された、状況に応じた対応への期待もこれまで以上に高まっています。すると、顧客との関係の強さはカスタマージャーニーの最も弱い部分によって決まってしまうことになります。

今の時代に顧客からの期待に応えるには、カスタマージャーニーを通じて使用される多数の異なるアプリケーションを統合して、クロスアプリケーションのデータ共有を促進する必要があります。統合戦略が成功すれば、多次元のカスタマーインサイトを生成して顧客のニーズを予測し、チャーンを最小化できます。

Uber はその好例です。Uber は破壊的デジタル革新の例として随所で取り上げられていますが、その成功に統合が与えた影響については見逃されがちです。タクシー会社は 20 年以上にわたって同じ顧客データを利用できる立場にありましたが、これまでは、複数のアプリケーションを連動させてデータを活用し、予測性を生み出して顧客の期待を変化させることはできませんでした。アジャイル・インテグレーションによってこのギャップが解消され、現在のデジタル世界でのイノベーションを生み出すほぼ無限のチャンスが生まれ、企業が競争する方法が変革されました。

現在の最も先進的な市場リーダー達はすべてをつなげて考えています。それは、デジタル・トランスフォーメーション、ひいては最終的な成功にとってインテグレーションが欠かせないと知っているからです。

1 Michael Bender および Paul Wilmott、Digital McKinsey、「Digital reinvention: Unlocking the 'how'」(2018 年 1 月) https://www.mckinsey.com/~/media/McKinsey/Business%20Functions/McKinsey%20Digital/Our%20Insights/Digital%20Reinvention%20Unlocking%20the%20how/Digital-Reinvention_Unlocking-the-how.ashx

「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。併せて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」[2]

The Phoenix Project
Gene Kim 氏

アジャイル・インテグレーション：柔軟性とスピードの獲得

スピードはデジタル世界にとって欠かせません。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必要があります。価格設定を変更したり、一晩で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。

デジタル経済で求められるスピードでソフトウェアを提供するには、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、柔軟で適応性が高く、より迅速に動作できることを指します。[3]

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を目標としていました。DevOps[4] 手法は、それをアプリケーションのデプロイにも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、それらを統合します。

Red Hat ではこのプロセスを「アジャイル・インテグレーション」と呼んでいます。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、開発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

Red Hat の CEO であった Jim Whitehurst (ジム・ホワイトハースト) はかつて次のように発言しています。「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。よくわからない環境での計画は効果的ではありません」。[5] 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。しかし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。アジリティとは、計画をすばやく策定し、すばやく実行に移す能力です。この環境では計画のスパンが短く、新しい計画が継続的に策定されます。

---

2 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

3 オックスフォード英語辞典

4 DevOps について理解する、https://www.redhat.com/ja/topics/devops

5 Jim Whitehurst の Red Hat Summit 2017 における基調講演。https://www.youtube.com/watch?v=8MCbJmZQM9c

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。

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新しい方法で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」[6] 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ゼロから構築します。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。

6 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

アジリティの基礎を築く

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。

1. 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
2. ユーザーが機能を迅速にアップグレード、追加、および削除できるテクニカル・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[7]、新しいプロジェクトとイノベーションを最優先事項とするアプローチです。

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。

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。このアプローチ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。

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

図 1. 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 3 つの柱は、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。

第 1 の柱：分散インテグレーション

従来の 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。このために、これまでは、ますます複雑で集中化したインテグレーションハブが生み出されてきました。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。

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。

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。

「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とき、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」[8]

「Continuous Delivery: Reliable Software Releases Through Build, Test, and Deployment Automation」
David Farley 氏

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。このアプローチで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[9]、開発速度を向上させ、リリースサイクルの迅速化を支援します。

この分散型アプローチにより、柔軟性が最大になります。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できるようになります。

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・アーキテクチャを分散することにより、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とき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。

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

ライフサイクルのステップ	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

分散バックプレーン上のストリーミング

アジャイル・インテグレーションでは、アプリケーション固有のニーズに応じて、同期または非同期の統合を自由に選んで使用できます。分散インテグレーションの一般的なアプローチでは同期手法を使用し、前述したように、コンテナをデプロイして異なるユーザー間でデータを共有します。分散インテグレーションのもう 1 つの手法は非同期手法であるストリーミングで、アジャイル開発者が必要に応じて他のソースからイベントを受信、再読み込み、再生できます。データはメッセージングを使用して中間ストアに複製されるので、複数のチームやアプリケーション間で共有できます。中間データストアを使用すると同期するデータを他のソースから探し出す必要がなくなるため、マイクロサービスチームにとってメリットがあります。

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

9 Martin Fowler 氏によるマイクロサービスのわかりやすい定義を参照：https://martinfowler.com/articles/microservices.html

分散データストリーミング・プラットフォームは、レコードのストリームをリアルタイムで公開、サブスクライブ、保存、処理できます。複数のソースからデータストリームを処理し、複数のコンシューマーに配信することに特化しており、プラットフォームをストリーミングすると、毎秒数百万のデータポイントを処理できるため、IT 運用や e コマースなど、リアルタイムでデータを取得する必要があるユースケースにおいては特に大きなメリットになり得ます。以下の図は、ストリーミングの多数のユースケースを代表するパターンを示しています。どのアプリケーションでも、運用すれば結果として変更が生じます。これらのアプリケーションでは、この変更をイベントとして 1 つまたは複数のストリーミング・プロセッサーに伝搬できます。その後、ストリーミング・プロセッサーはパターンマッチングまたは変換を実行し、現在の状況を過去の情報と照合します。この過去の情報は多くの場合、分析システムから提供されます。

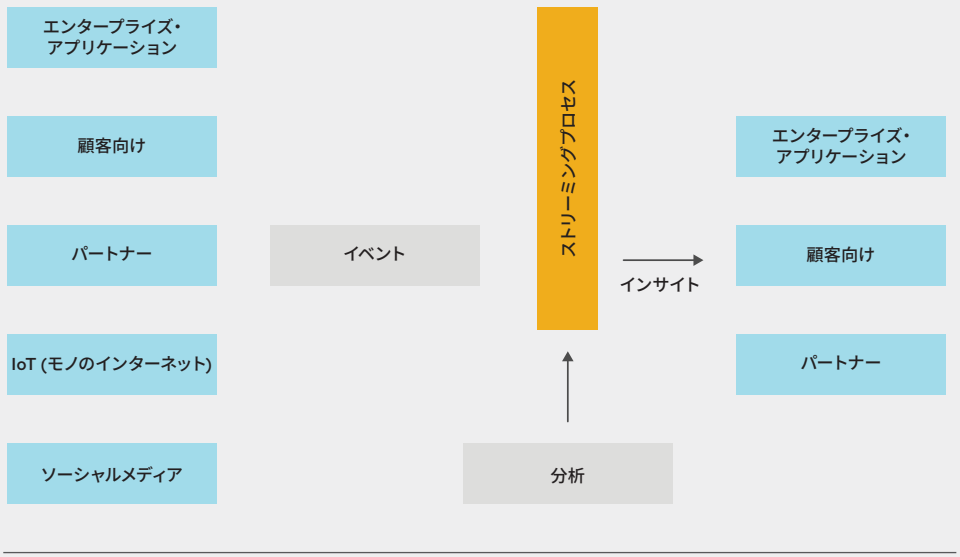

図 2. 処理プラットフォームからインサイトの生成までのイベントフロー

一部のユースケースでは、非同期統合のほうが同期統合よりも多くのメリットがあります。同期統合では、通信を成功させるには受信側が通信可能な状態でなくてはなりません。それに対してメッセージングベースの非同期統合では、受信側がリアルタイムで受信できなくても、メッセージを送信することができます。同期通信では、動作を続行するためにアプリケーションが応答を待機しなければならないので、タイムアウトがつきものです。この遅延は非同期メッセージでは発生せず、応答を受信したかどうかにかかわらず、処理が続行されます。こうした柔軟性により、ストリーミングなどの非同期統合は、大量のデータを処理する際に最適です。非同期統合ではシステムがタイムアウトする可能性が低いため、高可用性と信頼性も実現します。

ストリーミングなどのイベント駆動型非同期統合を使用して同期統合と API を補強すると、アジャイル・インテグレーションがさらに強化されます。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、高スループット、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。

イベントメッシュ

もう 1 つの新しい非同期分散インテグレーションはイベントメッシュで、分離されたアプリケーション、デバイス、クラウド間でイベントを分散させるための、構成可能な動的インフラストラクチャ・レイヤーです。

イベントメッシュは相互接続されたイベントブローカーのネットワークで構成され、イベントベースの非同期統合を処理します。環境に依存しないため、オンプレミス、プライベートクラウド、パブリッククラウド、ハイブリッドクラウド、PaaS (Platform-as-a-Service)、さらには IoT (モノのインターネット) など、デプロイ先がどこであっても、あるアプリケーションのイベントをその他の任意のアプリケーションに、イベントルーティングを構成することなく、転送して受信させることができます。

さらに、イベントメッシュによってマイクロサービスやクラウドネイティブ・サービス、レガシーアプリケーション、デバイス、データベースなど、あらゆるものを接続できます。基本的に、接続するイベントプロデューサーとイベントコンシューマーに制限はありません。イベントメッシュは非常に効率的で、イベントプロデューサーとコンシューマーとを結ぶ最速のパスを決定し、リアルタイムのインタラクションを可能にします。

イベントメッシュは分散化が進む IT 環境において、イベント通信の柔軟性、信頼性、速度、セキュリティを強化する先進的ソリューションとなります。

非同期統合とイベント駆動型アーキテクチャパターンは新しいものではありませんが、イベントメッシュは画期的な新アプローチで、インテグレーションの分野に登場したばかりです。イベントメッシュは現在広く普及しているわけではありませんが、今後デジタル・トランスフォーメーションによってイベントベースの統合が持つ柔軟性は企業になくてはならないものとなっていくため、Red Hat では今後この技術が牽引力を増していくと予測しています。

イベントメッシュはサービスメッシュとは違います。イベントメッシュは非同期ですが、サービスメッシュは API に基づく同期統合をサポートします。

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 AMQ とバンドルされ、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扱う場合に便利です。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。

Red Hat AMQ は、フォーチュン 100 のほぼ半数によって信頼されている[10] エンタープライズ Kubernetes プラットフォームの Red Hat OpenShift® Container Platform 上の AMQ Streams を介して、分散ストリーミング・プラットフォームである Apache Kafka を提供します。AMQ Streams は、Apache Kafka プロジェクトに基づく、非常にスケーラブルな高パフォーマンス分散データストリーミング機能です。この組み合わせにより、マイクロサービスおよびその他のアプリケーション間で、高スループットと低レイテンシーでデータを共有できます。

Red Hat が提供するイベントメッシュには、グローバルなピアツーピア・イベント・デリバリー・システムである AMQ Interconnect を介するものもあります。AMQ Interconnect は分散トラフィックマネージャーとして機能し、ボトルネックや障害を回避してルーティングして、シンプルな方法でコンシューマーに確実に到達します。この方法で、ノードとクラウドの障害に対するレジリエンシー (回復力) を実現します。

第 2 の柱：アプリケーション・プログラミング・インタフェース (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もののインタフェースとなって、この課題を解決します。

組織が、インテグレーションを集中型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の柔軟性を実現し、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。

API は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、開発者にとって統合の共通言語および統合を設計するための参考資料となります。

10 Red Hat プレスリリース、「世界中の 1,000 社以上の大企業がビジネス・アプリケーションの実現に Red Hat OpenShift Container Platform を採用」(2019 年 5 月 8 日)。https://www.redhat.com/ja/about/press-releases/more-1000-enterprises-across-globe-adopt-red-hat-openshift-container-platform-power-business-applications?extIdCarryOver=true&sc_cid=701f2000001OH74AAG

開発者はAPI をプロジェクトの構成要素として使用しますが、API は共有するために作られるものでもあります。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。たとえば、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が、適切な API を必要に応じて各グループに公開できます。

API をうまく活用するには、組織に API 管理機能が必要です。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。

サービスメッシュ

サービスメッシュは API 統合をさらに 1 歩進めたもので、マイクロサービス向けの構成可能なインフラストラクチャ・レイヤーを提供し、通信の柔軟性、信頼性、速度、管理性を実現します。

最初にマイクロサービスを使用するとき、組み込みのガバナンスがサービス間通信を処理し、サービス中断を最小限に抑えます。しかし新規のアプリケーション、サービス、機能がマイクロサービスとして継続的に追加されると、アーキテクチャの運用が複雑になり、ある時点で問題が発生します。新規サービスが追加されるたびに、障害点となりうる箇所が増えていき、サービスが要求で過負荷になりかねません。数百あるいは数千のマイクロサービスが相互に絶えず接続を試みると、通信遅延やアプリケーションのダウンタイムが発生します。そのうえ、複雑なマイクロサービスのアーキテクチャでは、問題の発生源を突き止めるのはほぼ不可能になります。このような問題に対処するために、サービスメッシュは導入されました。

サービスメッシュは専用のインフラストラクチャ・レイヤーで、アプリケーションに直接組み込まれています。サービスメッシュにより、サービス間通信を規定するロジックが個々のサービスから取り除かれ、インフラストラクチャのレイヤーに抽象化されます。サイドカープロキシはマイクロサービスと並んで存在し、リクエストを他のプロキシに転送します。これらのサイドカーがまとまってメッシュネットワークを形成します。このようにして、サービスメッシュは要求を他のサービスに転送し、すべてのマイクロサービスが最適に連携できるようにします。

サービスメッシュにより、ルーティング、フォールトトレランス、セキュリティ、可視性、監視、テストなどの追加機能をマイクロサービスに導入でき、マイクロサービス・コンポーネント自体には変更を加える必要はありません。この能力は、情報と機能をマイクロサービスに挿入する、サイドカープロキシによって実現します。

サービスメッシュにより、通信障害のトラブルシューティングが容易になりますが、これは問題の発生源がマイクロサービス内に隠されることがないからです。発生源は、可視化されたインフラストラクチャ・レイヤーにサービスとともに公開されます。

また、サービスメッシュによってアプリケーションが強固になり、ダウンタイムが発生しにくくなります。これは、要求が障害が発生したサービスに転送されないようにサービスメッシュが調整できるからです。

サービスメッシュはパフォーマンスメトリクスの収集も行い、サービス障害の発生時の診断機能を強化し、ダウンタイムを大幅に短縮できます。これと同じパフォーマンスデータを開発者が使用して、将来のリリースで設計を最適化できます。

サービスメッシュがなければ、各マイクロサービスはサービス間通信を可能にするロジックをコーディングする必要があり、これには開発者の時間と労力がかかります。サービスメッシュによってこのような余分なコーディングが不要になるので、開発プロセスが効率化されます。そこで、Red Hat ではこのテクノロジーが今後数年間で急速に普及すると予測しています。

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 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。

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、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。

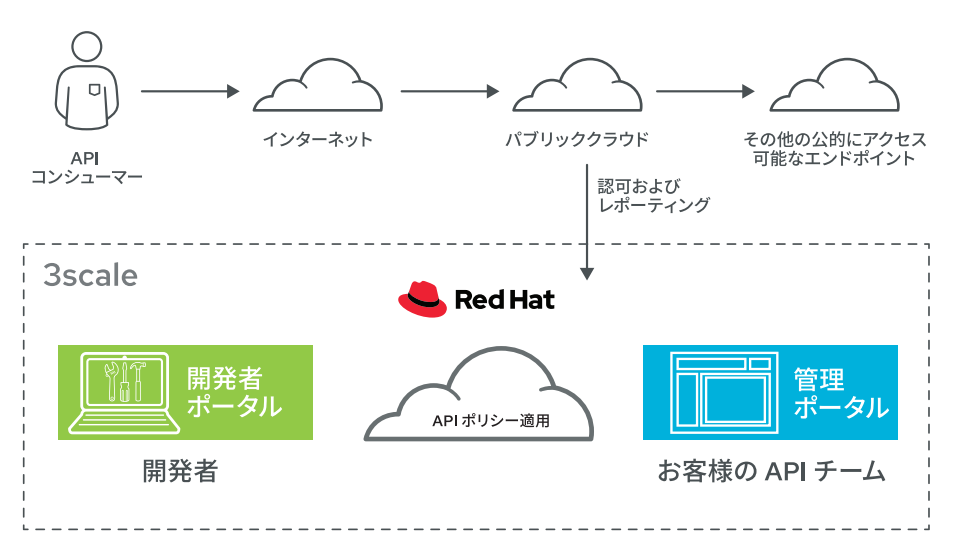

図 3. API 管理、エンドポイント、パブリッククラウドの概念図

Red Hat では Red Hat OpenShift Service Mesh も提供しており、人気のある Istio サービスメッシュと、トレース用の Jaeger や可視化用の Kiali などのその他の主要なプロジェクトを組み合わせ、マイクロサービスのデプロイの管理と追跡を向上させます。

さらに、3scale から Kubernetes の Istio サービスメッシュに対して、3scale Istio アダプターを使用して API を管理するための全機能が提供されます。

「マルチクラウドを実現するには、プロセス、スタッフのスキル、組織構造について熟考することになりますが、マルチクラウドを実現する技術プラットフォームがコンテナと Kubernetes であるという点では、業界の見解は一致しています」[11]

Forbes
Suresh Vasudevan 氏

第 3 の柱：コンテナ

仮想化、クラウド、コンテナの各テクノロジーは、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し、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するという、類似した目標を達成します。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。このユニットはインスタンスごとに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。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。

さらに、コンテナイメージは単一の機能ユニット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のビジョンを満たし、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。

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。

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、それらのサービスは、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。

11 Vasudevan、Suresh、「Containers And Kubernetes Are Powering The Second Cloud Adoption Cycle」、Forbes (2019 年 7 月 10)。https://www.forbes.com/sites/forbestechcouncil/2019/07/10/containers-and-kubernetes-are-powering-the-second-cloud-adoption-cycle/#171ce5006929

Red Hat OpenShift は Linux® 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。

アジャイル・インテグレーションの実装

チーム体制

アジャイル・インテグレーションの 3 つの柱は、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

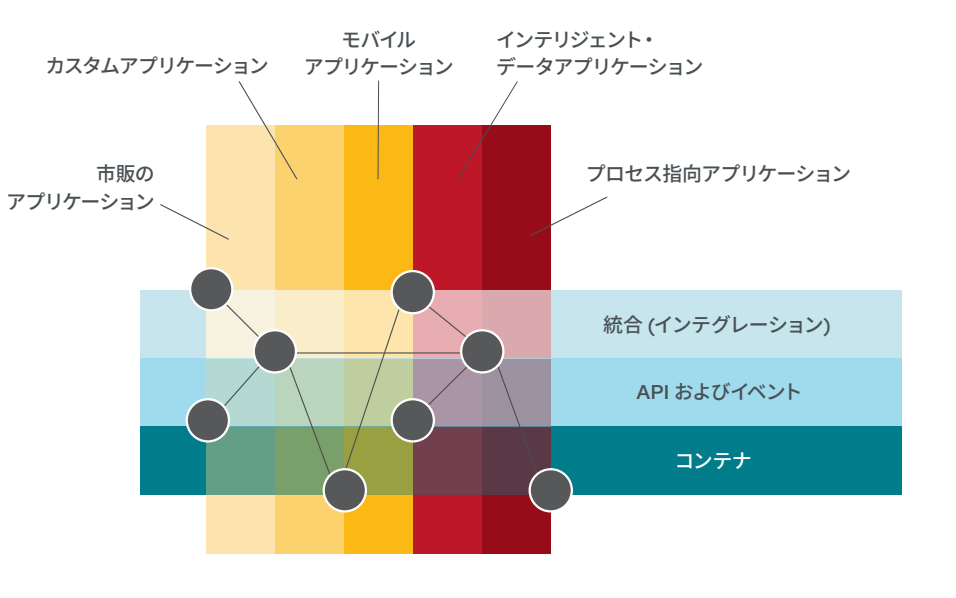

図 4. アジャイル・インテグレーション・テクノロジーとアプリケーションスタックの相関図

組織の内部ソフトウェアエコシステムは、多くの場合外部統合のアクセスポイントですが、同期レイヤーと非同期レイヤーの 2 つのレイヤーで成り立っています。同期レイヤー上では、コンテナ、API、統合が連携します。非同期レイヤー上では、イベント、コンテナ、統合が連携します。イベント処理には、イベントメッシュと分散ストリーミング・プラットフォームが使用されます。

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。

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。ソフトウェア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、プロジェクトが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 Froehlich 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。[12]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。しかし、効率性を最大にするためには、インフラストラクチャチームが開発チームおよび運用チームと連携することが重要です。グローバルコンサルティング会社の McKinsey は、「アジャイルな変革を使用して IT インフラ組織をモダナイズすることは容易ではないが、その価値があります。経験上、アジャイルなアプローチにより、IT インフラストラクチャ・グループはその規模に応じて、6 - 18 カ月のうちに生産性を 25 - 30% 向上できます」と結論しています。[13]

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。

---

12 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

13 Cormella-Dorda、Santiago, et al.「Transforming IT infrastructure organizations using agile」McKinsey Digital (2018 年 10 月) https://www.mckinsey.com/business-functions/mckinsey-digital/our-insights/transforming-it-infrastructure-organizations-using-agile

Agile Manifesto は、ソフトウェア開発の中核となる 4 つの原則を定義しています。[14] アジャイル・インテグレーションをベースにしたインフラストラクチャでは、これらの原則を統合戦略に適用できます。

1	**プロセスやツールより個人と対話を重視** インフラストラクチャでは、チーム間の相互作用を重視します。相互作用には、API、メッセージング、トラフィックパターンによって管理される直接通信、システムレベルの相互依存、CI/CD パイプラインなどのテストおよびリリースプロセスなどがあります。
2	**包括的なドキュメンテーションよりもソフトウェアが機能することを重視** インフラストラクチャは、元来、大きな変更を伴わず徐々に適応しながら、24 時間年中無休で機能するものでなければなりません。その意味で、機能するインフラストラクチャであることが常に暗黙の必須要件になっています。インフラストラクチャ戦略として、「機能する」とは、インフラストラクチャ・コンポーネントによって、予想されるパフォーマンスのエンベロープで、エンドユーザーが想定する動作をすることを意味します。
3	**契約内容の交渉よりも顧客とのコラボレーションを重視** インフラストラクチャ・システムのコントラクトでは、セキュリティポリシー、サービスレベル契約、さらには公開された API などのシステムの依存関係を、インフラストラクチャチームがどのように管理するかを規定します。顧客とは、これらのシステムの内部ユーザーと外部ユーザーの両方を指します。アジリティを取り入れると、これらのユーザーからシステムに関連付けられたポリシーおよびインタフェースの変更につながる意見を集め、それらの変更をより迅速に実行できるようになります。分散インテグレーションを使用すると、チームはインテグレーションの開発とデプロイメントを直接制御できるようになり、コラボレーションが拡張します。
4	**計画の遂行よりも変化への対応を重視** この原則では、テクノロジーがプロセスをサポートします。インフラストラクチャの場合、システムは安定した状態を維持する必要がありますが、コンテナなどの新しいテクノロジーは弾力性のあるプラットフォームを提供します。要求に応じたインスタンスの動的な追加と削除、デプロイメントと更新の自動化、複数のインスタンス間での変更の調整が可能です。公開された API 定義によって、開発の一貫性を高めるために再利用可能なツールが提供されます。このアプローチにより、変化に適応できる安定したプラットフォームが作成されます。

図 5. 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。

14 Agile Manifesto、http://agilemanifesto.org/

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、始動、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。Project Management Institute が実施した調査では、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。その調査ではこれを、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しています。[15]

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。

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。

15 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

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、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、つまり分散インテグレーション、API、コンテナというアジャイル・インテグレーションの 3 つの柱を使用して俊敏性と適応性の高いインフラストラクチャを作成する方法をアジャイル・インテグレーションと定義しています。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
#F20123_1119