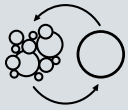
ホワイトペーパー

古いアプリケーションを生まれ変わらせる

継続的なアプリケーション・モダナイゼーションのための ソフトウェア開発のパターンとプロセス

Zohaib Khan



「現実を認めよう。
今開発しているのは、
明日のレガシー
ソフトウェアであることを」

MARTIN FOWLER 氏
THOUGHTWORKS チーフアーキテクト

facebook.com/redhatjapan
@redhatjapan
linkedin.com/company/red-hat

jp.redhat.com

エグゼクティブサマリー

過去数十年にわたる IT とソフトウェア・エンジニアリングを振り返ってみると、1 つのことが明らかです。それは「すべてが変化する」ということです。ハードウェア、言語、インフラストラクチャ、手法は段階的に改善されてきており、その中でも時折パラダイムシフトを引き起こすイノベーションが起こります。

この進化により、IT は絶え間なく変化するビジネス需要に先回りして対処できるようになりましたが、それは簡単でも安価でもありませんでした。多くの IT 予算は古いシステムの維持に回され、アップグレードや移行により最新のものについて行くだけで、ビジネスで利益が出る前に資金やリソースを使い果たしてしまう場合があります。

適切なアプローチをとれば、迅速かつ低コストで価値を生み出す方法でアプリケーションのポートフォリオをモダナイズすることができます。それにより、製品やテクノロジーが進化する中でも、より容易かつ低コストで、時代に合った最新の状態を維持することができます。

このホワイトペーパーでは、既存アプリケーションをモダナイズするためのソフトウェア開発パターンを 3 つ具体的に取り上げます。これらのモダナイゼーション・パターンでは、既存アプリケーションをより先進的なアーキテクチャやインフラストラクチャに移行させ、新しいアプリケーションからアクセスできるようにします。また、このホワイトペーパーでは、どのような場合に最終手段としてアプリケーションを書き換えなければならないかについても検討します。これらのパターンは、企業が既存アプリケーションを最大限に活用し、継続的なモダナイゼーションによって現在と将来の両方にわたってビジネスを成長させるためのプラクティスを確立する方法を知るうえで役立ちます。

アプリケーション開発およびデプロイメント

比較的最近まで、アプリケーションはプログラミング言語でコーディングされ、プロセッサーやオペレーティング・システムに固有のフォーマットにコンパイルされていました。アプリケーションは一般に自己完結的で、大規模になりがちで、プライベート・データセンターで実行されていました。誰もが、これらは長期間にわたって使用されるものと考えていました。アプリケーションは、形式の整った、事前に決められた要件に基づいて長い期間をかけて開発するという、大がかりなソフトウェア開発ライフサイクル・アプローチを使用して構築されました。

しかし、これらはすべて変わりました。このようなアプリケーションは、今では「モノリシックなレガシー・アプリケーション」と呼ばれ、ビジネス・アプリケーションの世界では化石のような存在となってしまいました。これらのアプリケーションは構築された目的を果たしたものの、ビジネスと技術におけるイノベーションの加速に伴い、企業にとっての負担となりました。

そして、このイノベーションにより、今日一般的に使用されるようになったアプリケーション開発とデプロイメントのモデルが生まれました。それは、クラウド上で実行されるコンテナにデプロイされるマイクロサービスの作成をガイドする DevOps プロセスです。アプリケーション開発の 4 つの分野、つまり手法、アーキテクチャ、デプロイメント、インフラストラクチャにおける進歩を考えてみましょう。

開発プロセスは以前よりずっと速くなっています。開発プロセスは、事前の要件定義と仕様決定からリリースまでの長い時間を要するウォーターフォール型の手法から、頻繁かつ段階的に機能のリリースを行う反復的な手法に発展し、さらに自動化された継続的インテグレーションとデリバリーによる高度にコラボレーティブな DevOps プラクティスへと発展しました。

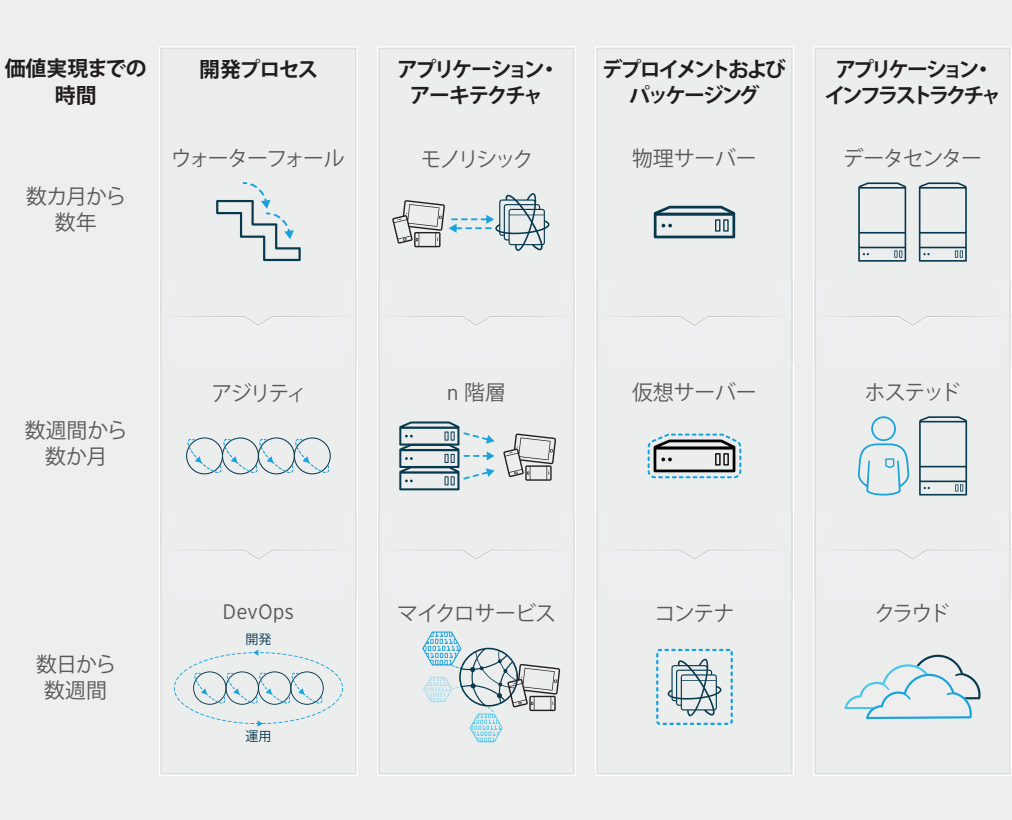


図1:アプリケーションの開発とデプロイの進化。

ソリューションスタック全体には、アプリケーションの開発およびデプロイに必要なすべてのコンポーネント、開発に使用する手法、およびそのアプリケーションが実行されるハードウェアが含まれます。

アプリケーション・アーキテクチャのトレンドは、機能をコンポーネントへと分割することです。オールインワン・アプリケーションは、インタフェース、ロジック、およびデータという別々の層に分離されました。サービス指向アーキテクチャにより、共通のエンタープライズ・サービス・バスを介し、専用サービスを使用してアプリケーションを構築することが可能になりました。この流れは、マイクロサービスと API に基づいた現在のモデルにつながりました。このモデルでは、高度に特化されたコンポーネントに、他のサービスやアプリケーションもアクセスすることができます。マイクロサービスと API は、インタフェース (どのようにアクセスされるか) と実装 (どのように実行するか) を区別する従来の Web サービスの標準に基づいており、様々な言語で実装し、異なるシステムにデプロイすることができます。

アプリケーションのデプロイメントは、はるかに柔軟になっています。アプリケーションは、もはやハードウェアに縛られることがありません。Enterprise Java™ などの標準に準拠して開発されるため、さまざまな組み合わせのハードウェアやオペレーティング・システムにデプロイできます。仮想マシンとコンテナにより、アプリケーションをパッケージ化して異なるホストにデプロイすることも容易になりました。

アプリケーション・インフラストラクチャは、大規模なアプリケーション固有のサーバーから、多数のアプリケーションをサポートする水平方向にスケーリングされたコモディティサーバーへと進化しました。現在では、さまざまな場所にあるデータセンター、プライベートクラウドおよびパブリッククラウドの複数のサーバーにアプリケーションをデプロイするのが一般的です。これにより、デプロイがはるかに迅速になり、パフォーマンスと可用性も向上します。

これらの 4 つの分野にわたるアプリケーション開発とデプロイメントの進化により、初期開発の迅速化、更新頻度の向上、品質向上、ビジネスニーズへのより緊密な適応、運用の柔軟性の向上、コストの削減が実現しました。

モダナイゼーションとは、新しいテクノロジーやプラクティスを採用することではありません。むしろ古いものをどう扱うかが重要です。

モダナイゼーション

最近では、アプリケーションをサポートするソフトウェア・コンポーネントをスタックと呼びます。広い意味でのスタック全体には、アプリケーションの開発およびデプロイに必要なすべてのコンポーネント、開発に使用する手法、およびそのアプリケーションが実行されるハードウェアが含まれます。

新しいプロジェクトには新しい製品や技術が採用されるようになりましたが、レガシー・ソリューション・スタックはいまだに多くの企業で使用されています。

たとえば、多くの金融サービス会社は数十年前にカスタム・アプリケーションを開発しました。そして、Bloomberg 端末とトレーダー・ワークステーション、クライアント・サーバー・システム、多層 Web アプリケーションが導入されました。現在、これらと同じ企業が顧客と従業員向けにモバイル・アプリケーションを開発しています。そして、事業成長や買収の結果として、アプリケーション・ポートフォリオ全体で数十のモダンスタックとレガシースタックが稼働している状況が見られます。

モダナイゼーションとは、新しいテクノロジーやプラクティスを採用することではありません。むしろ古いものをどう扱うかが重要です。たとえば、ある古い家を思い浮かべてみてください。家の古い部分の暖房には石炭を、新築部分の暖房では灯油とガスを使っているとします。家全体をソーラー発電にアップグレードしたとしても、高価な割にメリットが見合いません。しかし、新たな増築部分でソーラーを使用することは理にかなっており、家の他の部分とあわせて、全体として破綻なく機能させることには価値があります。

アプリケーションのモダナイゼーションには 2 つの主な目的があります。既存の機能とデータを新しいアプリケーションで可能な限り活用する (古いアプリケーションから新しい価値を引き出す) ことと、新しいプロセス、製品、テクノロジーの利点を古いアプリケーションにもたらすことです。

モダナイゼーションへの適切なアプローチ、つまり適切な手法とパターンにより、企業は変化を受け入れてさらに能力を発揮できるようになります。

モダナイゼーションのための 3 つのソフトウェア開発パターン

次の 3 つのソフトウェア開発パターンは、アプリケーションをモダナイズするための統一的なアプローチを提供します。そのうち 2 つは、既存アプリケーションの寿命と有用性を強化し、ポートフォリオ全体を一度に変更しなければならない事態を避けることで、アプリケーションの段階的なリファクタリングとアーキテクチャの更新をカバーする第 3 のパターンの準備を整えます。モダナイゼーションのメリットを受けるために、アプリケーションを一度に書き換える必要はありません。モノリシックなアプリケーションであっても、一から書き換えることなくモダナイゼーションの利点を活用することができます。

リフト & シフト

「リフト & シフト」では、既存アプリケーションのパッケージ化およびデプロイ方法をモダナイズします。このパターンでは、既存のコンポーネントが最新のデプロイメント・プラットフォームにデプロイされます。良く知られた例はアプリケーション仮想化です。アプリケーションはオペレーティング・システムと共にパッケージ化され、専用のハードウェアではなく仮想マシンで実行されます。

リフト & シフトは、アプリケーション・アーキテクチャをモダナイズするためのものではありません。むしろ、企業がアプリケーションを最新のデプロイメント・プラットフォーム上で実行しながら、後でアプリケーションをリファクタリングする時間の余裕を得るためのものです。

リフト & シフトは、アプリケーションを最新の高速なハードウェアにデプロイしてパフォーマンスを向上させるためにも使用できます。最新のプラットフォームではデプロイプロセスがシンプルなため、アプリケーションの柔軟性が向上します。専用サーバーを廃止し、管理を一元化することで、運用コストも削減できます。

リフト & シフトの一般的な例として、以下のような状況が挙げられます。

> プレゼンテーション、ビジネスロジック、およびデータサービスを備えた 3 層アプリケーションの各層が、異なる Linux® サーバー上で実行されている。各サービスは、すべての構成とランタイムの依存関係を含むコンテナとして再パッケージ化される。コンテナは、パブリッククラウド上で動作する PaaS (サービスとしてのプラットフォーム) 環境にデプロイされる。
>
> J2EE 上に構築され、4 つの仮想マシン上で動作する CMS (コンテンツ管理システム) は、一連のコンテナとして再パッケージ化され、PaaS にデプロイされる。PaaS で動作するコンテナの利点に加えて、開発チームは継続的インテグレーションおよび継続的デプロイメントを含む統合された開発者エクスペリエンスの利点も活用できる。

これらの例において、アーキテクチャには変更がないものの、アプリケーションは最新のデプロイメント・プラットフォームに移し替えられ、再び活用できるようになります。これにより、アプリケーションをリファクタリングしてアーキテクチャをモダナイズするための時間が生まれます。たとえば、4 つの CMS コンポーネントをマイクロサービスとして再構築し、CMS 全体を API としてカプセル化することで、コンテンツ管理機能を必要とするモバイル、クラウド、その他のアプリケーションからアクセスできるように変更できます。

リフト & シフトは、すべてのアプリケーションに適しているというわけではありません。特定のベンダー・ソリューションに組み込まれているアプリケーションは、再パッケージ化や再デプロイが難しい場合があります。古いオペレーティング・システムが、最新のデプロイメント・プラットフォームでサポートされない場合もあります。

新規レイヤーによる拡張

多くの企業では、モバイルや Salesforce などの新しい経路でアプリケーションを提供したり、パートナーのアプリケーションと統合したりすることでビジネス価値を創造しています。「新規レイヤーによる拡張」は、既存アプリケーションの機能を新しいアプリケーションや経路からアクセスできるようにするソフトウェア開発パターンです。この場合、機能を開発し直す必要がないため、開発時間とコストを削減できます。すでにある複雑で重要な機能が、時間をかけて実績が証明されてきたのであれば、それを使用するほうが効果的であると言えます。

新規レイヤーによる拡張の手法では、アプリケーション・ソフトウェアの新しいレイヤーを作成します。このレイヤーは、既存アプリケーションの機能とデータを、新しいアプリケーションからアクセス可能なインタフェースで包む役割を果たします。過度な複雑化を避けるため、新しいレイヤーには通常余分なビジネスロジックを含めず、単純に新しいソフトウェアと古いソフトウェア間のアダプターとして機能させるようにします。

通常、既存アプリケーションを修正する必要はありません。そのため、このパターンはソースコードが利用できない場合、または既存アプリケーションを変更することのリスクが大きい場合に適しています。

リフト & シフトの場合と同様に、既存アプリケーションのアーキテクチャは変更されません。しかし、新規レイヤーによる拡張においては、既存の機能を使用して、現在のアーキテクチャを利用する新しいアプリケーションおよびサービスが作成されます。

新規レイヤーによる拡張は、アプリケーション・アーキテクチャのモダナイゼーションを段階的に進めるための準備となります。これにより、長期的には古いアプリケーション機能を書き換え、古いスタックをリタイアさせることが可能です。このパターンを使用して、(ストラングラー・アプリケーションと呼ばれる) 既存アプリケーションからゆっくりと移行することができます。これは通常、一度にすべてを書き換えるよりも優れたアプローチです。

新規レイヤーによる拡張の例を紹介します。

> 既存の商用アプリケーションが、現在の仮想化またはコンテナ・プラットフォームでサポートされていないオペレーティング・システム上で動作している。これには、標準 API を使用してアクセスできる。この API にアクセスするアダプター・マイクロサービスが作成される。このマイクロサービスは、新しいアプリケーションや他のマイクロサービスが商用アプリケーションの機能にアクセスするために使用される。
>
> Visual Basic で作成された発注アプリケーションが、データベース・バックエンドの複雑なストアード・プロシージャを大量に使用している。現在、携帯電話とタブレット向けの新しいモバイル発注アプリケーションの開発が進んでいる。新しいアプリケーション用のマイクロサービス・インタフェースを備えたアダプターレイヤーが作成される。アダプターによって、データベースとストアード・プロシージャーはアプリケーションから隠される。すべての新しいアプリケーションは、アダプターにアクセスする。アダプターは、その要求をストアード・プロシージャーの呼び出しに変換する。

どちらの例でも、新規レイヤーによる拡張は、現行のアーキテクチャでの新規開発の際に既存アプリケーションを使用できるようにするための暫定的なソリューションに過ぎません。このパターンはリフト & シフトとも矛盾がなく、同時に適用することができます。一方は既存アプリケーションの機能を新しいアプリケーションで利用できるようにし、他方は既存アプリケーションのデプロイと管理をより簡単かつ低コストに行えるようにします。どちらのパターンも、開発者が既存アプリケーションを現在のアーキテクチャでリファクタリングして書き換えるための時間を生み出します。

書き換え

アプリケーションを書き換えることは、一から新しいアプリケーションを作成することとは異なります。アプリケーションの書き換えとは、既存アプリケーションを置き換え、廃止するために新しい機能を作成するプロセスです。書き換えは、全体的なモダナイゼーション戦略の一環として、リフト & シフトと新規レイヤーによる拡張に続くステップになり得ます。これは、アプリケーション・アーキテクチャ全体を更新して最新のスタックを実現するための唯一の方法です。

既存アプリケーションの書き換えは通常、3 つ選択肢のうち最も魅力のないものです。費用と時間がかかる可能性が高く、費用が相殺されるまでに何年も要する場合があります。アプリケーションが新しいビジネス価値を提供するのでない限り、予算に関して厳しい経営幹部に書き換えの正当性を説明することは困難です。

しかしながら、書き換えしか選択肢がない場合もあります。たとえば、ベンダーによるサポートから外れてしまうオペレーティング・システムやハードウェア上で実行されている、非常に古いアプリケーションが存在する場合があります。企業は、ベンダーサポートというセーフティーネットなしで重要なビジネスシステムを実行したくないと考えます。また、古いシステムを操作する技術をだれも持っていないという場合もあります。

書き換えしか選択肢がない場合、最善のアプローチは古いアプリケーションから機能を徐々に移行することです。これは、新規レイヤーによる拡張を行うことで可能になります。また、中にはいずれ廃止され、そもそも移行の必要がなくなる機能もあるため、書き換えを遅らせることも良い考えです。単純に古い動作をそのまま移行したいと思うかもしれませんが、そうではなく、新しいアプリケーションを開発する場合と同じように計画し、優先順位を付けてください。これは、より柔軟で今後生じる変更に対応できるアプリケーションを作成するのに役立ちます。

パターンの選択

最善のパターンは 1 つではありません。アプリケーション、ビジネス、および背景的な要因によってさまざまに異なります。これらのパターンは、アプリケーションの開発とデプロイメントのさまざまな領域に対応しているため、複数のパターンが 1 つのアプリケーションに適用される可能性も、あるいはどれもまったくメリットがない可能性もあります。一般に、ポートフォリオをモダナイズするには、複数のソフトウェア開発パターンが必要です。

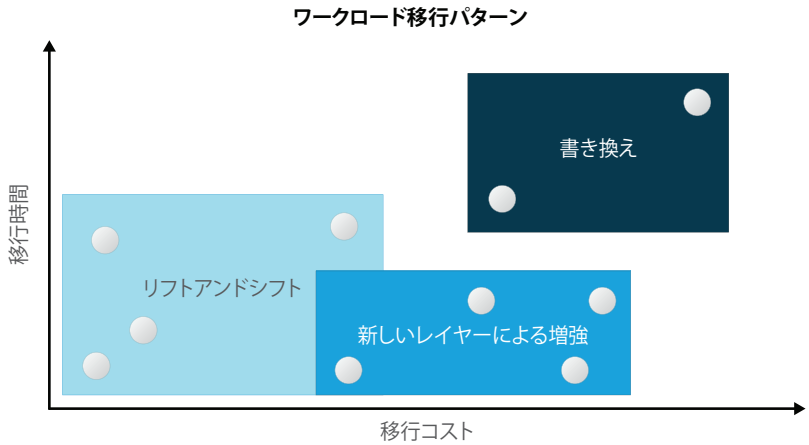


図 2：時間対コストの関係で決定されるワークロードの移行パターン。メリットはアプリケーションの現状と将来の状態によって異なるため、ここには表示していません。

リフト & シフトは、通常、適用に最も費用のかからないパターンです。新規レイヤーによる拡張は開発に手間がかかるためコストが高くなる可能性がありますが、多くの場合、迅速に適用することができます。書き換えは、ほとんどの場合、最も高価で時間がかかります (図 2)。

これらのソフトウェア開発パターンは、企業が既存アプリケーションを最大限に活用し、継続的なモダナイゼーションによって現在と将来の両方にわたってビジネスを成長させるためのプラクティスを確立する方法を知るうえで役立ちます。

モダナイゼーションの手法

これら 3 つのモダナイゼーション・パターンは、アプリケーション・アーキテクトや開発/運用管理者が既存アプリケーションに新たな命を吹き込むために役立ちますが、通常はいずれも、より大きな取り組みの一部に過ぎません。その場合、定まった手法があれば、努力を組織し、コストをチェックし、ビジネスに価値をもたらすうえで役立ちます。

Red Hat では、アプリケーションのモダナイゼーションに複数のフェーズからなる反復可能なプロセスを使用します (図 3)。主な目的は、計画を作成し、それを複数回にわたって実行し、実行のたびに段階的な成果を得ることです。これにより、廃棄やリプレースのリスクが軽減されます。

「発見」フェーズでは、インタラクティブなセッションにより、現在のアプリケーションの状態と目的を識別します。主要な利害関係者は、可能性のあるモダナイゼーションの方法を調べ、先に進めるためのコミットメントを得るという目標でステップの優先順位付けを開始します。

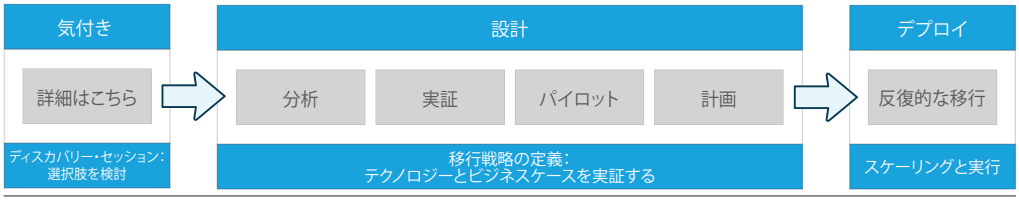


図 3 : Red Hat のアプリケーション・モダナイゼーション・プロセス

設計フェーズでは、一連の手順によりプロセスを分析から、概念実証、パイロットへと進めます。自動化されたアプリケーション・コード分析は、潜在的な問題を特定し、作業量を見積もるのに役立ちます。一方、ユーザーは現在の状態を識別してモダナイゼーションの候補を決定するために、スタックとアーキテクチャを詳しく調べます。目標とする状態はこのステップで設計します。ここでモダナイゼーションのパターンを選択し、詳細なデプロイメント計画を作成します。

次のステップで、デプロイメント計画を実行します。リスクを軽減し、以前の手順から学べるように複数の反復に整理されたこのモデルは、移行スペシャリストを擁するセンター・オブ・エクセレンスによって導かれる必要があります。プロジェクトの展開に伴い利害関係者は変わっていきますが、このグループがベストプラクティスを把握することで、持続可能性が確保されます。

まとめ

アプリケーションのモダナイゼーションを大規模なアプリケーションのポートフォリオに適用することはとても難しく思えるかもしれません。古いスタックや旧式のハードウェアでアプリケーションが実行されている場合はなおさらです。必要な時間と労力を見積もることは難しく、運用コストを引き下げるという理由は資金調達を正当化するのに十分ではありません。

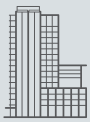
Red Hat の手法では、問題を分割することで、一度にいくつかのアプリケーションだけを移行します。アプリケーションは、モダナイゼーションに必要なコストと時間、モダナイズ後の運用コスト、潜在的なビジネス価値に基づいて評価されます。

3 つのモダナイゼーション・パターンは、既存アプリケーションの寿命を延ばし、いつ書き換える必要があるかを判断するのに役立ついくつかの標準的なアプローチを提供します。モダナイゼーションへの適切なアプローチ、つまり適切な手法とパターンにより、企業は変化を受け入れてさらに能力を発揮することができます。

Red Hat® コンサルティングは、このホワイトペーパーで紹介したパターンを使用して、組織が既存アプリケーションからより多くの価値を引き出せるよう支援します。詳しくはこちらをご覧ください。
redhat.com/ja/resources/application-migration-modernization-consulting-jboss-middleware-datasheet

執筆者紹介

Zohaib Khan は現在、Red Hat でアプリケーション移行プラクティスのリードおよび PaaS コミュニティ・オブ・プラクティスのマネージャーを務めています。Zohaib はこれまで、Red Hat Summit、StackWorld、Red Hat Technical Exchange など、さまざまなカンファレンスでプレゼンテーションを行ってきました。また、DevOps やテクノロジーによる破壊的創造とイノベーションについても執筆しています。Red Hat に入社する前には医療サービスおよび金融サービスのシニアマネージャーを務めていたほか、テクノロジーサービス企業を共同設立するという、起業家としての一面もあります。



RED HAT について

オープンソース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クラウド、Linux、ミドルウェア、ストレージおよび仮想化テクノロジーを提供、さらにサポート、トレーニング、コンサルティングサービスも提供しています。Red Hat は、お客様、パートナーおよびオープンソースコミュニティのグローバルネットワークの中核として、成長のためにリソースを解放し、ITの将来に向けた 革新的なテクノロジーの創出を支援しています。

アジア太平洋
+65 6490 4200

インドネシア
001 803 440224

ニュージーランド
0800 450 503

ベトナム
800 862 6691

オーストラリア
1 800 733 428

日本
03 5798 8510

フィリピン
800 1441 0229

中国
800 810 2100

ブルネイ / カンボジア
800 862 6691

韓国
080 708 0880

シンガポール
800 448 1430

香港
852 3002 1362

インド
+91 22 3987 8888

マレーシア
1 800 812 678

タイ
001 800 441 6039

台湾
0800 666 052

facebook.com/redhatjapan
@redhatjapan
linkedin.com/company/red-hat

jp.redhat.com
#046201_1216