



SUCCESS

エンタープライズDevOps

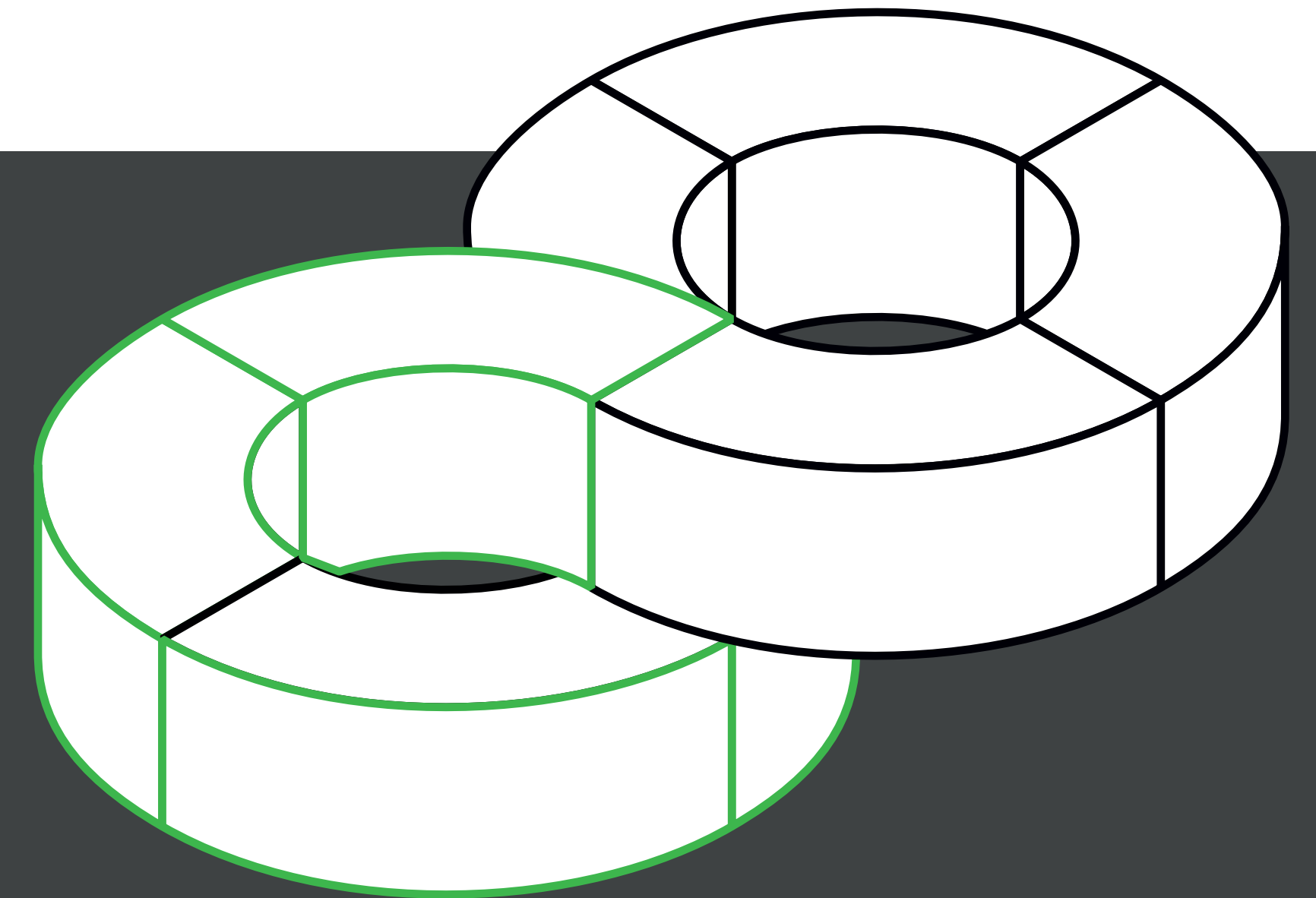
大規模なDevOpsで
成功するための5つの鍵

[DevOpsのメリット](#)を実感すると企業は自然にもっと大規模な環境で活用しようとしします。しかしチームがDevOpsへの取り組みを拡大しようとする、小規模なユースケースで機能しているツールやプロセスがうまく動かなくなることがよくあります。既存システムとクラウドの両方で、さまざまなチーム、ツールセット、アプリケーション、プロセス、ワークフロー、リリースサイクル、パイプラインをすべてサポートしなければなりません。そうしないと自動化サイロや[DevOpsツールとプロセス](#)が無計画にごちゃまぜになり、品質、セキュリティ、パフォーマンス、最終的には成功の度合いにも影響してきます。

今やソフトウェアが事実上すべてのビジネスを支え、促進しているため業界で成功することが生き残る上での必須の課題となっています。DevOpsはアプリケーション開発とデリバリの繰り返しの共同作業で、この新しいバリューチェーンの中心に位置しています。大規模なシステムを実装するには、適切な構造、プロセス、ツールが必要です。

本書では組織全体でDevOpsを効果的にスケールさせるための5つの主な原則について解説します。

- エンドツーエンドのDevOpsを集中管理 ▶
- 初期段階からセキュアにする方法 ▶
- クラウドによる将来的な保証 ▶
- パイプラインをコード化 ▶
- グローバルに考え、ローカルで行動する ▶



エンタープライズ向け DevOpsを成功させるための 5つの考え方

エンタープライズDevOpsのパイオニアでありリーダーでもあるJFrogは組織全体でDevOpsを効果的に拡張するために何が必要かを知っています。あらゆる業界の世界最大級の企業を含む6,000社を超える顧客を抱えるJFrogであるからこそ、大規模なDevOpsについて熟知しています。

これらの大企業でDevOpsが採用され、高品質のソフトウェアを頻繁かつ大規模にデリバリし、現在ではクラウドの近代化を進めています。ここでは企業がDevOpsを拡張する際に、私たちの顧客が実践している5つの主な基本方針について説明します。

1

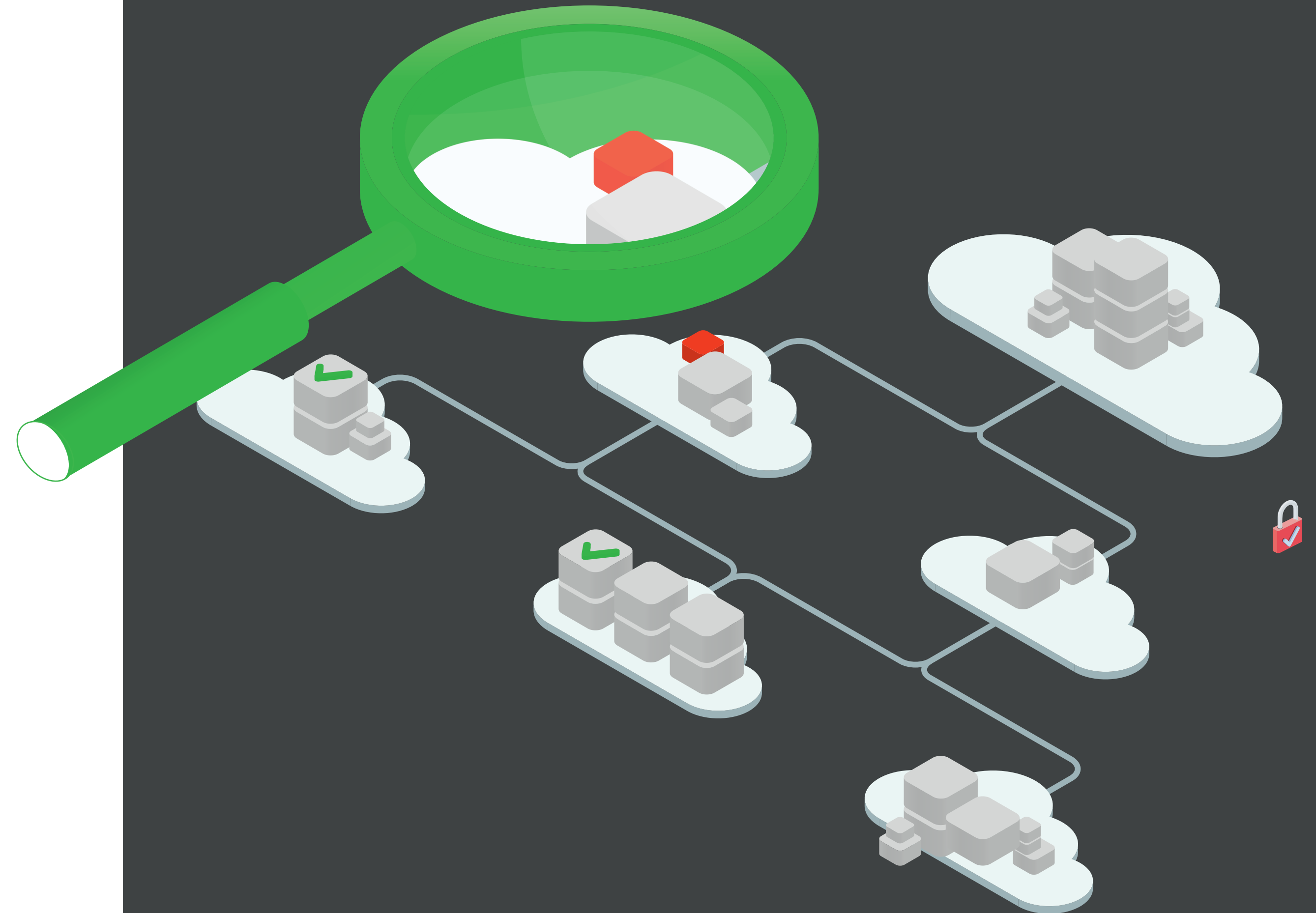
エンドツーエンドプロセスと そのアウトプットを一元管理する

エンドツーエンドの中心にある[DevOps Platform](#)には、バイナリ、[コンテナイメージ](#)、[CI/CDパイプライン](#)、[セキュリティとコンプライアンス](#)、ランタイム環境、エッジ、「モノ」にまたがる最終的なデプロイに至るまでの[ソフトウェア配布](#)の管理が含まれます。現在の多くの[CI/CDツール](#)は自動化プロセスまたはそのアウトプット(ビルドとバイナリ)のどちらかを管理することはできますが、両方を管理できないだけでなくすべてのバイナリタイプや技術をサポートしていません。

そこで一つのソリューションで[アーティファクトを一元管理](#)できるDevOps Platformが必要不可欠です。統一されたエクスペリエンスは[SDLC](#)全体とソフトウェア資産を明確に可視化し、信頼できる唯一の真のソースを提供します。これによりソフトウェアのデリバリーを高速化し、コードの品質、セキュリティ、運用が向上し、依存関係のダウンロード、リポジトリ、デプロイ、ビルド、パイプライン、リリースのすべてにおいて自動化されたプロセスの実行とトリガーが可能になります。

単一のエンドツーエンドDevOpsソリューションでデリバリプロセス、アセットやアウトプットの両方を管理し、異なるツール間で「コンテキストスイッチ」をする必要がないため次のことが可能になります。

- すべてのアーティファクトが開発から本番環境までのパイプラインを通るときに、ソフトウェアのライフサイクル全体で一貫性とトレーサビリティを確保します。
- さまざまなタイプのバイナリ、コンテナイメージ、環境、プロセス、ポイントツールなどのユニバーサルリポジトリと信頼できる唯一のソースを提供します。
- サードパーティを含むすべてのツール、プロセス、アーティファクトからリポジトリに至るまでセキュリティとコンプライアンスを管理し、組織全体の運用を確保します。
- パイプラインと組織全体の完全な可視化を実現します。サイロ化されたプロセスやスノーフレーク構成はもう必要ありません。



2

初期段階からセキュアにする — 組み込み開発と「シフトレフト」機能

エンタープライズDevOpsでは、開発からデプロイ、本番環境に至るまでのソフトウェアのライフサイクル全体でセキュリティとコンプライアンスのチェックを組み込む必要があります。アプリケーションコードの60%から80%は、サードパーティのオープンソースコンポーネント(OSS)で構成されています。アプリケーションの一部としてOSSを使用し、エコシステムで利用可能な既存のコンポーネントを利用することで、開発者の時間を節約し生産性が大幅に向上します。しかしながら、これらの依存関係にはしばしば[セキュリティの脆弱性や設定ミス、ライセンスコンプライアンスの問題](#)やその他の運用面での問題を抱えることになります。

今日の企業ではかつてないほどのソフトウェアの増加を管理しなければなりません。企業はますます多くのアーティファクトやアプリケーションを作成しており、そのすべてがOSSコンポーネントを使用しています。その上、[マイクロサービスベース](#)のクラウドネイティブアプリケーションの継続的な採用によって無数のサービスが接続され、各コンテナイメージには何百ものOSS依存関係をもつ何十ものレイヤーが含まれるため攻撃対象はさらに広がっています。

このシナリオは、悪意ある第三者の一步先を進むためには、今までより迅速にこのような脆弱性にパッチを適用することが急務であるにも関わらず、一般的な企業には200人の開発者に対してセキュリティ専門家が1人しかいない、という事実によってさらに悪化しています。

ソフトウェア開発のライフサイクルの最後にセキュリティテストを実施しようとする、それがボトルネックとなりデリバリーに遅延が発生します。セキュリティテストやスキャンは面倒なタスクなので、ボトルネックとなるプロセスの最後に回される傾向があります。

どのような解決策があるのでしょうか？

- セキュリティとコンプライアンスは選択したDevOpsプラットフォームの不可欠な部分で、デフォルトで有効にすべき一番の要です。もうツールやコンテキストを切り替えたり、テストの実行やスキャンの開始を忘れたりする必要はありません。
- セキュリティはユニバーサルで、コンテナイメージなどのクラウドネイティブアーティファクトを含むすべてのバイナリタイプをサポートし、アーティファクトのライフサイクルとCI/CDパイプライン全体で強固に統合されている必要があります。
- アプリケーションレイヤーからオペレーションシステム(OS)レイヤーまでのコンテナイメージを含むすべての依存関係を再帰的にディープスキャンする必要があります。開発者が手を煩わせることなく、OSレベルまで自動的にスキャンし、きめ細かい依存関係ツリーを特定できないようなツールには注意が必要です。
- スキャンはパイプラインでトリガーされるのではなく、すべてのリポジトリと本番環境のインスタンスにわたりDBレベルで継続的かつ自動化されている必要があります。なぜなら多くの依存関係がすでに「システム」の中にあり、既存のアプリケーションやビルドにまたがっている可能性があるからです。DBレベルでスキャンすることで、新たに発見された脆弱性に対するパッチを現在使用されている「アクティブな」CI/CDのデリバリパイプラインの一部ではない古いアプリケーションや再利用可能なパッケージに適用できます。DBレベルのスキャンに加えて、別の検証段階でCI/CD自動化の一部として、セキュリティ監査やセキュリティゲートをトリガーさせるべきです。
- 「シフトレフト」を有効にします。アプリケーションのセキュリティソリューションはIDEと統合して、アプリケーション開発プロセスのできるだけ早い段階で脆弱性を特定することでパッチを適用できるようにする必要があります。コードがビルドされるまで待つと本番環境に出荷するよりも、脆弱性を早期に発見すればするほど、その修正にかかるコストは低くなります。
- ソフトウェアの開発段階と自社のDevSecOps展開計画に応じて、実施する範囲とフォローアップアクションを柔軟に調整し、セキュリティとコンプライアンスポリシーの運用ルールを設定します。ポリシーはチーム、アプリケーション、ユースケース、DevSecOpsの成熟度やリスクに応じて厳しくしたり弱くできます。

3

クラウドネイティブによる将来的な保証： 近代化の必要性

Kubernetesのような最新のクラウドネイティブパターンとテクノロジーを活用するためにアプリケーションを最新化する場合でも既存のアプリケーションを更新できることと、すべてのアプリケーションがコンテナ化されたマイクロサービスであるとは限らないことを忘れないでください。

エンタープライズDevOpsプラットフォームはクラウドネイティブと既存アプリケーションの両方をサポートしどちらのタイプのアプリケーションのライフサイクル全体でも管理できる必要があります。これにはバイナリ、CI/CDプロセス、セキュリティスキャンなどが含まれ、別のツールに切り替える必要はありません。

同様に、DevOpsプラットフォームはハイブリッドとマルチクラウドの両方である必要があります。これによりDevOpsを利用してオンプレミス、プライベートとパブリッククラウド、マルチクラウド、エッジインフラのハイブリッド環境におけるデリバリパイプラインを管理できます。

4

Pipelines-as-Code (コードとしてのパイプライン)

Pipelines-as-Code (コードとしてのパイプライン)を定義することは開発者の生産性を高め、DevOpsへの取り組みを拡大するのに役立ちます。パイプラインの定義をソース管理システムに保存することでパイプライン定義を共有(チーム内のコラボレーションが可能)、バージョン管理、再利用、監査、再現が可能になります。

これにより開発者間での冗長な作業が減り(チームがCI/CD自動化を作成するため車輪の再発明する必要がなく)組織全体で精査された自動化プロセスを標準化して活用できます。

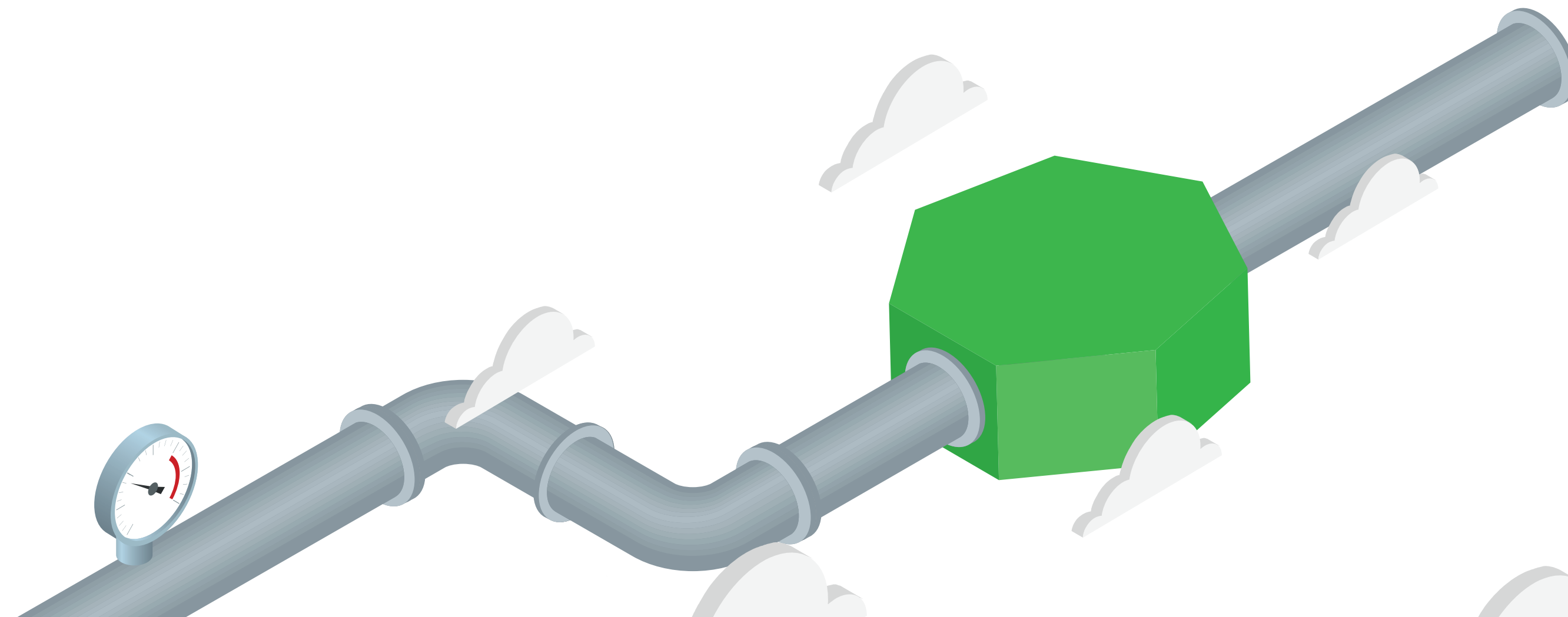
さらにチームが製品出荷のようなデリバリパイプラインにおいて、各バージョンごとにプロセスを強化し、改善し、強化することで継続的に進化させられます。

Pipeline-as-Codeのメリットとベストプラクティスを詳しく見ていきましょう。

■ チーム間での再利用と標準化:Pipelines-as-Codeを活用してDevOpsチームが組織全体のワークフローとプロセスを標準化するために必要な構成要素をモデル化して定義できます。これには主に2つのメリットがあります:

- オブジェクト、プロセス、シークレット、リソース、構成、ポリシー、セキュリティテスト、条件付き実行などの自動化要素を再利用や共有してスピードを早め、開発者の生産性が向上します。
- 組織全体で一貫性のある承認済みの強化されたプロセスを標準化することで品質と運用が向上します。これらの取り組みによって軌道のずれや、品質低下を引き起こす可能性のあるスノーフレーク構成、自動化サイロ、異種プロセスやエラーが発生しやすいスクリプトが排除されます。

- パイプラインのコードでパラメータを使用します。別々のチームで同じPipelines-as-Codeを利用するには、これらの構成要素をパラメータ化してすべてのチームが適切な値(パスワードなどのシークレット、リソース、環境設定パラメータなど)を動的に呼び出せる必要があります。
- 宣言型自動化を使用します。Pipelines-as-Codeの構成要素は「スパゲッティコード」と呼ばれるエラーが発生しやすいスクリプトや手間のかかる作業を回避して、より簡単に定義および拡張できるように宣言型であることが望ましいです。また、宣言型パイプラインはKubernetesのようなクラウドネイティブ環境にも適しています。
- 従来のワークフローを最新化します。大規模な企業には多くのレガシーなスクリプトや技術的負債を抱えている可能性が高いためレガシーなCI/CD技術(古いビルドツールなど)とカスタムスクリプト(PerlやBashの自動化スクリプトなど)を保護するために柔軟なDevOpsプラットフォームが必要です。これらは外部システムやカスタムコードを呼び出す標準ステップや統合することで、Pipelines-as-Codeスクリプトでもサポートしていく必要があります。これによりレガシースクリプトがリファクタリングや最新化されるまでの間、最新のCI/CDソリューションによってトリガーやオーケストレーションされ、エンドツーエンドの自動化を実現できます。
- DevOpsプラットフォームチームが組織でのDevOps採用を拡大し、認定プロセス、プロモーションゲート、セキュリティやコンプライアンスチェックなどに対応できます。



5

グローバルに考え、ローカルに行動するためのプラットフォームチームに贈る言葉

システムレベルで考えることはDevOpsの重要な理念です。これはDevOpsエコシステム全体とソフトウェアデリバリにおける実践方法(チーム、文化、バリューストリーム、プロセス、技術、アーキテクチャの選択など)すべてにおいて、体系的かつ全体的に考えることを意味します。

企業でDevOpsを拡張していくには組織全体のプロセスとツールを考慮する一方で、開発者に権限を与え、チームの自律性とスピードを実現するための柔軟性、敏捷性、選択の自由を考慮する必要があります。DevOpsプラットフォームは「グローバルに考え、ローカルに行動する」必要があります。例えば特定のユースケースにおいて、さまざまなセキュリティポリシーを適用したり、そのユースケースにあった最善のポイントツールを統合していく必要があります。

このアプローチにより方向性のずれを減らし、バラバラなワークフローや構成になることを防ぐ一方でレガシーとクラウドネイティブ両方のアプリのデリバリで、DevOpsエコシステムにおけるいかなるツールやプロセスもサポートするコンプライアンスと柔軟性を確保します。ここがDevOpsプラットフォームチームの見せ場です。

アプリケーションのデリバリは今日の企業競争で勝ち抜くために非常に重要です。デリバリの規模と頻度が増えるにつれ、多くの企業はさまざまなチームのアドホックやサイロ化されたソリューションでデリバリを管理し続けるのは非現実的であると考えています。



DevOpsプラットフォームチーム（プラットフォーム運用やデリバリサービスとも呼ばれます）は共通ツール、標準化されたプロセス、一貫性のある運用ですべての内部チームに対してDevOps-as-a-Serviceを有効にするツールとデリバリするインフラを選定する責任を負います。自動化された構成要素を開発、強化、認定してチームが一元管理されたりリポジトリから取得可能にします。新しい要件が発生したときには、これらのプロセスをさらに強化します。そのため以下の方法でスピード、生産性、信頼性、TCO（総保有コスト）を向上させます。

- Pipelines-as-Codeを活用し、組織でのDevOps採用を拡大・加速すると同時に運用、コンプライアンス、監査可能性を確保します。
- セルフサービスのノードのプルやシークレットローテーションのカatalogを管理し、開発者が安全な方法で一貫した構成で環境をプロビジョニングできるようにします。
- 開発者が利用する「認定済み」パッケージの共有リポジトリへ中央アクセスを可能にします。プラットフォームチームはOSSの依存関係を強化および保護し、[組織の運用ポリシー](#)に準拠していることを証明します。これにより手直しが不要になり、共有された安全なアーティファクトの再利用と標準化が可能になります。
- パイプラインとソフトウェアデプロイのチームやステージ間で一元管理され、デプロイの最終段階に向けたバイナリ共有を管理し、スループットやネットワーク使用率を最適化し、セキュリティや運用を確保します。
- アクセス制御（RBAC：Role-based Access Control）を一元管理し、リポジトリ、アーティファクト、ツール、パイプラインプロセス、承認、環境など承認されたすべてのコンポーネントへのアクセスレベルを包括する機能を備え、コンプライアンスと監査性を確保します。

JFrog Platformは世界中の大企業で使用され、品質とセキュリティを向上させながらアプリケーションの配布を高速化するエンドツーエンドのDevOpsソリューションです。JFrog PlatformはSDLCをエンドツーエンドで合理化、保護、拡張し、バイナリ管理、コンテナイメージ、CI/CD、アプリケーションセキュリティ、配布までサポートします。

JFrog Platformのクラウド版は無料でお試しいただけます。



JFrog Japan 株式会社

〒 100-0004 東京都千代田区大手町1-9-2 Global Business Hub Tokyo | TEL: 03-4243-1049 | Webサイト:jfrog.com/ja/ | ブログ:jfrog.com/ja/blog/
お問い合わせ:jfrog.com/ja/contact-us/

- JFrogの名称、ロゴマークおよびすべての JFrog製品の名称は、JFrog Ltd.の登録商標または商標です。
- その他、本書に記載されている会社名および製品・サービス名は、各社の登録商標または商標です。
- JFrogは、通知を行うことなく、いつでも該当製品およびサービスの提供、機能を変更する権利を留保し、本書中の誤植または図表の誤りについて責任を負いません。