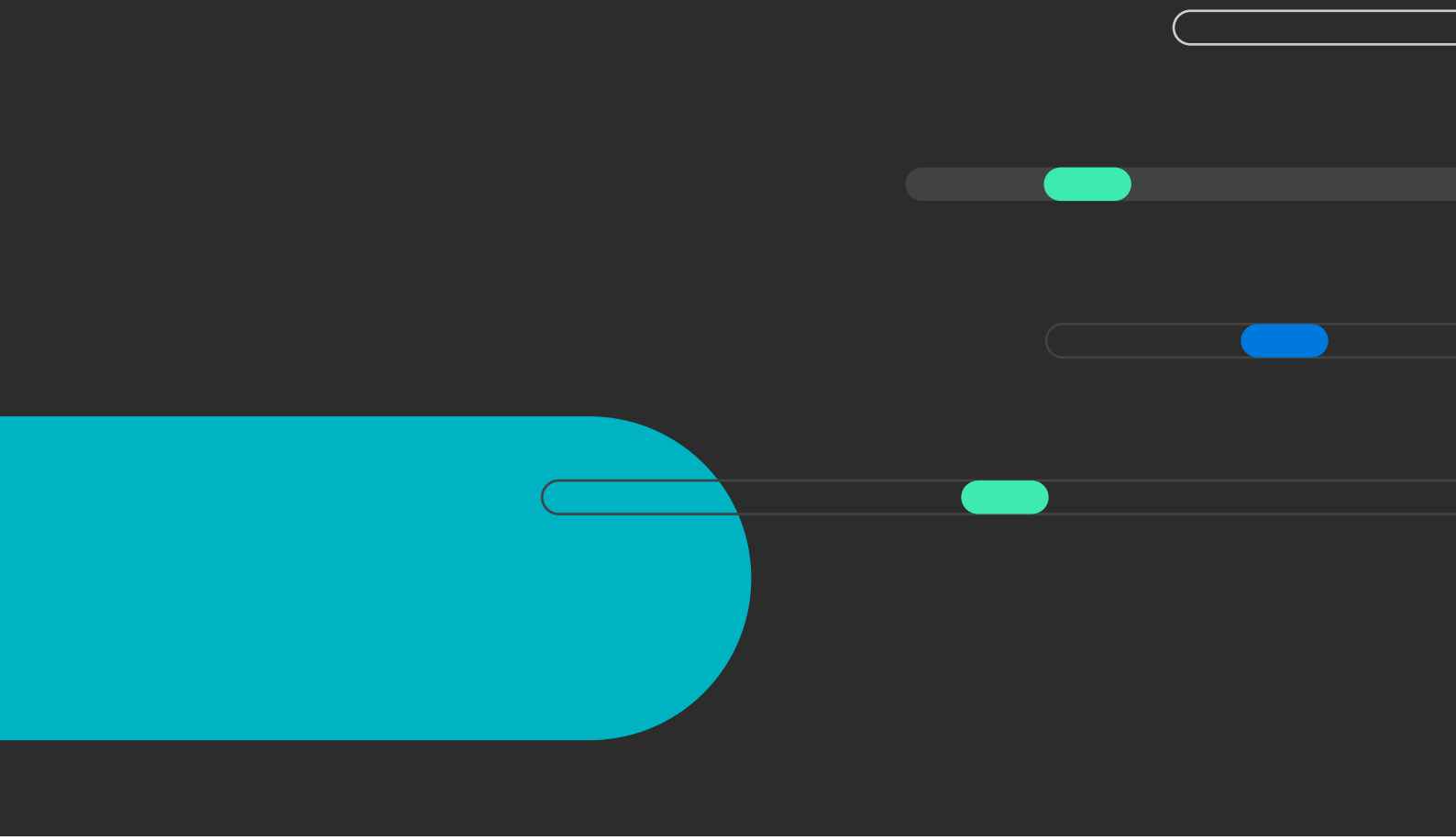
New Relic

ホワイトペーパー



DevOpsの次のフェーズへの準備

最新のクラウド環境の複雑さを軽減するための3つの方法

目次

はじめに：津波のように押し寄せる複雑さ

New Relicでは長年にわたり、組織がDevOpsを適切な方法で遂行できるよう支援してきました。この間、私たちはDevOpsに関する独自の経験を共有し、チームがDevOpsとクラウドネイティブテクノロジーを最適に活用するための新機能を継続的に提供し、DevOpsの成功を測定、追跡するためのデータ駆動型のアプローチとベストプラクティスを主導してきました。

DevOpsが主流となり、それを採用した組織が繰り返し成功を収めている今、DevOpsとクラウドネイティブのテクノロジーやテクニックがもたらす深刻な影響が出てきていることを認識すべき時がやってきました。マイクロサービス、サーバーレス、コンテナ、その他のアプローチやテクノロジーをクラウドで採用した結果、複雑さという津波が組織を襲い始めています。

DevOpsとクラウドネイティブのアプローチが悪いというわけではなく、逆に両者のメリットは、複雑さから生じるコストをはるかに上回っています。重要なのは、複雑さがリスクを生み、それがITとビジネスの成果に悪影響を及ぼす前に、複雑さを軽減するための措置を組織が講じる必要があるということです。DevOpsに取り組んでいる企業が、最新のアプリケーション開発の次のフェーズに必要なスキル、ベストプラクティス、ツールをまだ習得していない場合は、アプリケーションのパフォーマンスと可用性の目標を達成し続けることが極めて困難であることに気づくはずです。

このホワイトペーパーでは、現在のDevOpsの状況、複雑さが増している理由、そして組織が複雑さを早急に軽減する方法について説明します。

DevOpsの背景をもっと知りたいですか？

このホワイトペーパーでは、DevOpsの基本的な理解を前提としていますが、DevOpsのジャーニー半ばでも役に立つ追加のリソースをご紹介します。

- What Is DevOps?（DevOpsとは）
- DevOps Done Right: Best Practices to Knock Down Barriers to Success（正しいDevOps：成功への障壁を打ち破るベストプラクティス）
- DevOps Without Measurement Is a Fail（測定なきDevOpsは失敗に終わる）

多大なメリットをもたらしたDevOps

DevOpsの原則、文化、プロセス、ツールは、今日の組織がソフトウェア主導の世界で直面する課題、すなわちデジタル化、革新、市場投入時間の短縮、変化への迅速な対応、および完璧なカスタマーエクスペリエンスの提供の推進に役立ちます。

DevOps Research and Assessment（DORA）は「State of DevOps 2019」レポートの中で、「多くのアナリストが、DevOpsとテクノロジーの変革に関して業界が『キャズム（深い裂け目）を越えた』と報告していますが、今年の私たちの分析は、その見解を裏付けています。業界の加速度は増す一方で、速度と安定性の両立が可能となった今、クラウドテクノロジーへの移行がこの加速に拍車をかけています」と述べています。

DORAのレポートでは、クラウドが速度と安定性を促進するとされていますが、その分析によると、クラウドコンピューティングの使用は、ソフトウェア配信のパフォーマンスと可用性の予測にもつながることがわかっています。

DevOpsにおける役割

DevOpsにおける役割と責任は元々、もっと明確なものでした。

- **Dev:** エンジニアは、インフラストラクチャ、プロビジョニング、容量など、アプリケーションをエンドツーエンドで管理します。
- **Ops:** エンジニアは、モニタリングや自動化/スクリプト作成など、開発者向けツールを使用して、インフラストラクチャを管理します。

今日、DevOpsの目標は「顧客への強化機能の出荷を妨げる組織的な障壁をすべて取り除き、ツールと自動化によって加速すること」に進化したため、役割も曖昧になっています。

DORAのレポートで最も高いパフォーマンスを記録したチームは、アメリカ国立標準技術研究所（NIST）が定義したクラウドコンピューティングの5つの機能すべてにおいて、低いパフォーマンスを記録したチームと比べて、24倍も高いパフォーマンスを達成しました。

たとえば、AWSのクラウドコンピューティングでは、エンジニアがAWS Service Catalogを通じて、インフラをプロビジョニングするためのセルフサービスメソッドを構築できます。開発者は、サービスがプロビジョニングされるまで待たずに、新しいことをすばやく試し、フェイルファスト（早期に失敗）して、新しい製品をより早く市場に投入することができます。完全なマネージドサービスによってAWSリソースの活用が支援され、AWSのデベロッパー用ツールによる自動化によって、より迅速かつ効率的にリソースを活用できます。

DevOpsが適切に機能すれば、ITとビジネスの両面で大きな成功を収めることができると言っても過言ではありません。しかし、水平線上には暗雲が垂れ込めています。バックラッシュ（反動）が起きようとしているのでしょうか？

成功の代償

スピード、アジリティ、回復力、スケーラビリティなど、最新のアプリケーション開発の中核となるメリットを享受するために、DevOpsチームは代償を払い始めています。その代償とは、複雑さです。モノリシックアプリケーションが複雑でないわけではありませんが（脆弱で扱いにくく、拡張が難しいなど、一般的にイノベーションを遅らせる原因となる特性もある）、最新のクラウドネイティブ環境は、別の種類の複雑さを生み出しており、それに対応する準備ができていないチームも存在します。

チームは、モノリシックアプリケーションを、独立して所有、展開できる小さなマイクロサービスに分割することで、モダナイズしています。マイクロサービスは保守が容易であり、再利用することで開発を加速させることができます。チームは同時に、Amazon Elastic Container Service（ECS）とAmazon Elastic Kubernetes Service（EKS）を使用するコンテナ（Docker）とコンテナオーケストレーション（Kubernetes）、またはAWS Lambdaを使用するサーバーレスなどの最新テクノロジーを採用することで、アプリケーションの移植性と拡張性を高めています。

Kubernetesクラスタエクスプローラに表示されている特定のポッドのKPI

これらのことから、数百のコンテナで実行されたり、サーバーレス機能として動作したりするマイクロサービスが増加しています。その上、毎日、あるいは毎時間、多数の異なるサービスが同時にデプロイされています。そこに、コンテナ、負荷分散、自動スケーリングなど、流動的なインフラストラクチャと自動化が組み込まれています。

その結果、大量の可動部分が刻々と成長し、変化していくのです。問題の根本的な原因を探るには、一体どこを確認すればいいのでしょうか？特定の状況下で短時間しか存在しないコンテナで問題が発生しているのかもしれません。

複雑さを生み出すのは可動部分だけでなく、それぞれの部分から発生する大量の動作ノイズによっても生み出されます。インフラストラクチャのあらゆる部分とあらゆるマイクロサービスが、メトリック、イベント、ログ、トレースに関するデータを生成する可能性があります（このテレメトリデータのコレクションをM.E.L.T.と呼びます。このトピックの「Introduction」で詳しく説明しています）。パフォーマンス上の問題の真の原因を探るには、どこを確認すればいいのでしょうか？

複雑さを軽減し、実用的な分析情報を失うことなく、ノイズを調整することが、DevOpsジャーニーの次のステップになるはずです。

DevOpsの次のステージに向けた重要な原則

チームはこれまで、DevOpsの「Dev」の部分に多くの労力を傾けてきました。つまり、ソフトウェアの信頼性と可用性を高め、エラーを減らして、より速く、より頻繁に提供する能力を高めてきました。

これからは、DevOpsの「Ops」の部分に注力しましょう。最新のクラウドネイティブアプリケーションと環境の複雑さを軽減するためには、2つの重要な原則が役立ちます。

1. 文化を正しく理解することに重点を置く

DevOpsを成功させるためには、これまでも、そしてこれからも、文化が重要です。文化とは、チームがどのように形成され、チームメンバーがどのように協力して作業するか、そして構築されるソフトウェアに対する責任をどのように分担するかということです。チーム内およびチーム間の透明性とコミュニケーションは、DevOpsの目標をサポートするための共通の理解を深めるのに役立ちます。

> **「マイクロサービスという新しいソフトウェア構築方法は、迅速かつ大規模な変化に合わせて最適化されますが、その代償として複雑さが生じます。組織が分散システムに伴う複雑さに対応する準備ができていない場合、Kubernetesのような強力なテクノロジーでは、真のリスクが発生する可能性があります。お客様には、最新のシステムに対応するための準備を本格的に行い、最新のツールを使って作業していただくようお願いしています」**
>
> Amazon Web Services DevOps担当、
> SIプラクティスリード Kelly Looney氏

ただし、「信頼」という核心的な特性は、時に見落とされてしまいます。信頼は、DevOpsチームが受け入れ、習得する必要のある最も重要な文化的要素でしょう。リーダーはチームを、チームは他のチームを、そしてチームメンバーはお互いを信頼し合わなければなりません。

信頼がなければ、自律性や責任という他の本質的な文化的特性は機能しません。出荷する製品を所有する自律性と責任、そしてそれを正しく行うために必要なツール（Amazon Elastic Beanstalk、AWS Lambda、AWS CloudFormationなど）をチームに与えたら、正しく動いてくれるはずだとチームを信頼しなければなりません。

2. あらゆるものをインストルメント化し、測定する

測定は、DevOpsの専門家であるJez Humble氏が考案したCALMSフレームワーク（Culture（文化）、Automation（自動化）、Lean（リーン）、Measurement（測定）、Sharing（共有））の5つの柱の1つです。複雑さが増す中、DevOpsを成功させるためには、包括的でコンテキストに対応するデータが不可欠です。このような複雑な状況に対処するためには、アーキテクチャがどれほど一時的なものであっても、その全体像を把握することが重要です。適切なデータがあれば、何がうまく機能し、何がうまく機能していないのか、そしてチームをどこに集中させればいいのかがわかります。

データは、意思決定プロセスから感情を排除することで責任の所在を明らかにする一方、コラボレーションと共感の文化を育みます。これにより、DevOpsチーム内の全員に、スキル、経験、役割を超えた共通言語が提供されます。

Netflixが「Operate-What-You-Build（開発したものが運用する）」モデルを採用

Netflixは、同社のTech Blogの記事で、ソフトウェアを構築したチームがその運用とサポートにも責任を持つ、共有オーナーシップ方式に移行することを決定した経緯と理由を語っています。

複雑さを軽減し、成果を継続的に向上させる3つの方法

DevOpsの取り組みを成熟させ、進化させ、次の複雑なステージに備えるために、Opsのパフォーマンスを向上させながら、チームが習得に努めるべき3つのコンピテンシーがあります。

1. オブザーバビリティ

最新のクラウドネイティブ環境における複雑さの解決策は、オブザーバビリティです。オブザーバビリティは、あらゆるものをインストルメント化し、テレメトリデータを使って、システム内の関係や依存関係、パフォーマンスや正常性に関する実用的な基本情報を取得することで実現できます。

オブザーバビリティを確保できれば、チームは、接続されているすべてのパフォーマンスデータのコンテキストビューを1つの場所でリアルタイムに確認し、問題を迅速に特定し、問題の原因を理解し、最終的に優れたカスタマーエクスペリエンスを提供できます。これは、チームが変更についての十分な情報を得た上で意思決定するのに役立ちます。チームが更新するサービスは、その変更によって破損する可能性のある他のサービスと通信しているでしょうか？

オブザーバビリティについて詳しくは、「The Age of Observability: Why the Future Is Open, Connected, and Programmable（オブザーバビリティの時代：未来を担うのはオープン、コネクテッド、プログラマブルである理由）」をご覧ください。

---

2. 高速フィードバックループ

失敗を成功させること、つまり「フェイルファスト」は、DevOpsに望まれる能力です。著者、研究者、兼DevOpsのエキスパートであるGene Kim氏は、DevOpsのプロセス、プロシージャ、プラクティスを構成する3つの方法の1つとして、増幅されたフィードバックループを挙げています。そのためには、逆算式のフィードバックループを作り、それを短縮、増幅して継続的に修正することが重要であると述べています。

高速フィードバックループは、信頼の原則と、チームが以下のことを実行できるようにするツールやプロセスを組み合わせたものです。

- **実験を奨励する：**イノベーションと大胆なアイデア（失敗も含む）の文化を推進し、市場投入までの時間を短縮し、最新のテクノロジーの採用を加速させ、ビジネスの成果を向上させます。
- **頻繁にデプロイする：**マイクロ機能やバグ修正といった小規模なデプロイメントを頻繁に行うよう奨励し、各デプロイメントの成功を測定します。パイプラインのインストルメント化について詳しくは、こちらのウェビナーをご覧ください。
- **リスクを受け入れる：**小さくて影響の少ない失敗は、より速く出荷するためのコストだと認識すればいいでしょう。完全性よりも一貫性が重要であることを理解した上で、本番環境でより小さな問題がより頻繁に発生することを受け入れます。本番環境の問題が発生しても、全体として見れば、大規模な停電に比べれば、その影響ははるかに小さいでしょう。
- **あらゆる失敗から学ぶ：**インシデントを解決した後、正確かつ徹底した文書化を進めることで、チームはインシデントから学び、再発を防ぐための予防策を講じることができます。そのためには、懲罰や責任追及ではなく、建設的な学習と改善に焦点を当て、責任追及をせずに振り返ることが望ましいと考えられます。
- **リリースダッシュボードを使用する：**機能リリースごとに、その機能に特化した主要業績評価指標を追跡する「機能ダッシュボード」を作成します。ダッシュボードでは、アプリケーションへの影響を理解するために、機能のデプロイメントと、その機能によるAWSサービスの使用状況を追跡する必要があります。

特定の機能のデプロイメントに関連するKPI

3. 継続的な改善

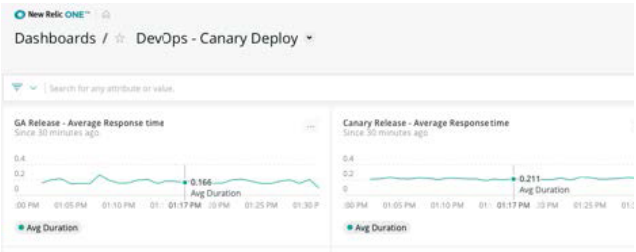
DevOpsを採用してから、複雑さを軽減し、成果を最適化するハイパフォーマンスのDevOps組織になるには、継続的な改善に専念する文化とプロセスが必要です。DORAの「State of DevOps」レポートで、エリートパフォーマーの数が前年比3倍になったのは、こうした継続的な改善の文化が育まれているからです。

まず、パフォーマンスとビジネスの成果を関連付けることから始めましょう。これにより、インフラストラクチャとアプリケーションの変更が、最終的にカスタマーエクスペリエンスとビジネスメトリックにどのような影響を与えるかを明らかにすることができます。その結果を測定値として示すことができなければ、デプロイメントが成功したと主張するのは難しいでしょう。

複雑さが生じていてもDevOpsのパフォーマンスを向上させ続けるための方法として、高度なリリース戦略を利用するという方法もあります。

- **副作用のないサンドボックスを利用できるようにする：**実験を奨励するためには、リスクを取って新しいアプローチやテクノロジーを試しても影響が及ばない環境を用意する必要があります。
- **機能フラグをオンにする：**リリースがあると、機能フラグがオンになります。機能が期待どおりでない場合は、オフに切り替えることもできます。
- **サービスの利用者とやり取りをする：**各ビジネスユニットは相互に、本番レベルの利用者の役割を果たす必要があります。これは、新バージョンと旧バージョンを同時にメンテナンスする期間だということです。その結果、オーバーヘッドが増えることになりますが、悩みの種が生じない低リスクのリリースが可能になります。
- **ロールアウトのアプローチをとる：**リスクの高い機能については、ユーザー全体にロールアウトする方法をとります。機能フラグは、トグルではなくメーターでも構いません（例：ユーザーの0% → 5% → 10% → 25% → 50% → 70% → 100%）。

- **カナリアテストを使用する：**カナリアテストとは、コードの変更を一部のユーザーに公開し、大多数のユーザーが利用し続けている古いコードと比較して、どのように実行されるかを調べる手法です。カナリアサーバーやコンテナで新しいコードが実行され、新規ユーザーが検出されると、そのサブセットがカナリアホストに転送されます。New Relicは、新しいコードが正常に機能しているかモニタリングし、測定するのに役立ちます。



カナリアのデプロイのKPI

- **ブルーグリーンデプロイメントを採用する：**本稼働時のダウンタイムを回避し、重大な問題が発生した場合のロールバックを迅速化するために、ブルーグリーンデプロイメントアプローチの使用をぜひ検討してみてください。ブルーグリーンデプロイメントでは、ほぼ同じ本番環境を2つ用意し、1つをグリーン、もう1つをブルーとします。ブルーの環境を現在の本番環境とした場合、グリーンの環境では、最終テストを実施し、その成功に基づいて、本稼働時にユーザーをその環境へルーティングするようにします。ブルーの環境はその後アイドル状態になりますが、必要であれば迅速にロールバックできます。

- **ダークローンチを利用する：**ダークローンチはカナリアテストに似ていますが、新機能をローンチして、そのコードが旧機能と比較してどのように実行されるかを評価するのではなく、少数のユーザーにリリースして、新機能に対するユーザーの反応を評価する手法です。この手法では新機能を強調しないため、通常はユーザーに気づかれることがありません。それがダークローンチという言葉の由来です。

たとえば、Webサイトで新しいUIをダークローンチするには、別のドメインやURLでローンチするか、ホームページの非表示のIFrameに含めます。

責任追及をしない

責任追及を伴わない振り返りの目的は、問題の再発を回避する、あるいは少なくとも軽減することを目標として、すべての関係者が、インシデントに至った状況、結果として生じるインシデントの対応、プロセス改善のギャップやポイントについて理解を深めることです。

このブログ記事では、責任追求を伴わない振り返りを実施する方法とその理由について説明しています。

まとめ

DevOpsで「キャズムを越える」組織が増えるにつれて、不可欠となるのは、複雑さを軽減し、DevOpsの「Ops」部分の改善に集中的に取り組むことです。チームをハイパフォーマーやエリートパフォーマーのカテゴリに引き上げるには、オブザーバビリティを構築し、高速フィードバックループを活用し、継続的に改善する必要があります。

New Relicは、最新のアプリケーション環境の複雑さを解消するのに役立ちます。AWSとの緊密な統合により、EC2、Lambda、Kubernetesのデプロイメントなど、複雑なAWS環境の管理を支援します。

詳しくは、newrelic.com/jp/devopsをご覧ください。

New Relic.