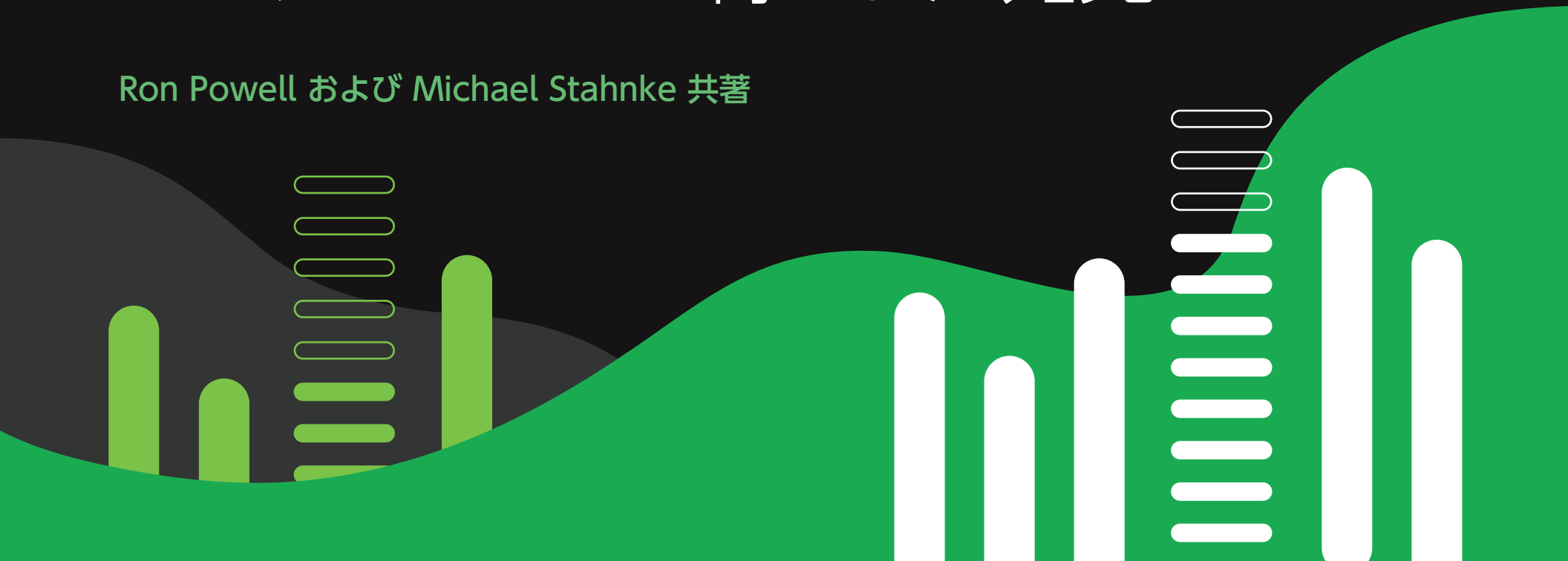


データで見る CI (継続的インテグレーション) を 活用しているチームの利点:

DevOps の実践に関する 3000 万の
ワークフローから得られた知見

Ron Powell および Michael Stahnke 共著





要約

優れたパフォーマンスを発揮している、テクノロジーデリバリーチームは実際、どのように稼働しているのでしょうか？自社のチームが適切に稼働していることをどのように把握できるでしょうか？テクノロジーデリバリーチームの行動についての調査結果がこれまでも多く報告され、優れたチームのメトリクスは定義されていますが ([Puppet State of DevOps Report 2019](#)、[2019 Accelerate State of DevOps](#))、テクノロジーデリバリーチームの実働状況に関する極めて大規模なデータを確認できるアドバンテージが CircleCI にはあります。CircleCI のクラウド CI（継続的インテグレーション）および CD（継続的デリバリー）プラットフォームは、4 万社以上の組織と 15 万件以上のプロジェクトで、1 日あたり 160 万件以上のジョブを処理しています。CircleCI では、3,000 万のワークフロー* データを分析し確認した状況と報告されている業界標準のデータを比較しました。

CircleCI の以下の調査結果から、継続的インテグレーションが優れたパフォーマンスを発揮するチームに不可欠であることが分かりました。

- CI を使用するチームは非常に高速にジョブを処理している。すべてのワークフローの 80% が 10 分未満で終了している。
- CI を使用するチームはフローに従って作業を継続して実行している。すべての復旧の 50% は 1 時間以内に行われている。
 - 25% は 10 分で復旧している。
 - 50% の組織は 1 回の処理で復旧している。
- 効率的な CI ツールを使用することで、ビジネスニーズに対応することが可能になっている。パイプラインが柔軟であり、パイプラインの変更が成功している。90 日間の調査期間中、パイプラインが変更されたとしても 50% のプロジェクトが失敗することがなかった。
- **エンジニアリングを成功させるための手法を検討されている場合、CI に着目することで優れた成果が得られる。**

CircleCI を使用しているチームは、自信を持って高速なデリバリーを実現しています。これは、優れたパフォーマンスを発揮している開発チームの要件です。**CI は自信と自動化をもたらします。**

CircleCI によって優れたパフォーマンスを発揮するようになるのか、それとも優れたパフォーマンスを発揮しているチームが CircleCI を選んでいるのかという疑問はまだ決着していませんが、**CI の実践に取り組んでいるチームは、いつでも自信を持って、最高のソフトウェアを迅速にデリバリーしていることは明らかな事実です。**

CI を確実かつ広範に導入して実践することは容易ではないことを CircleCI は理解しています。また優れたソフトウェア製品をデリバリーすることも容易ではありません。しかし、CircleCI のデータには素晴らしい結果が示されており、CI を使用している平均的なチームでも、最高レベルのテクノロジーデリバリーチームにランクインしています。最高の開発チームになるために、世界最高の DevOps の実践者を採用する必要はありません。1 日に 1,000 回デプロイしたり、インフラストラクチャに毎月 100 万ドルを費やしたりする必要もありません。**CI/CD の実践に着手するだけで、エンジニアリングを成功させることが可能です。**

* ワークフローは、ジョブのコレクションと実行順序を定義する一連のルールです。

はじめに：

テクノロジーデリバリーチームであれば、コードを迅速にデリバリーし、市場にリリースするコードの品質を向上したいと考えているでしょう。DevOps における業界標準の「4 つの主要なメトリクス」は、ソフトウェアの開発とデリバリーのデジタルトランスフォーメーションを推進し、スピードを犠牲にすることなく、安定性を最適化できることを示しています。しかし、これらのメトリクスから開発チームが実際にどのように稼働しているのかを把握するにはどうしたら良いのでしょうか？開発チームがさらに優れた成果を達成できるようにするために、これらのデータから実用的な知見をどのように得ることができるのでしょうか？

CircleCI がこの調査を実施した理由は、これらの疑問への答えを把握したかったためです。世界最大のスタンドアロン CI プロバイダーとして、CircleCI はプラットフォームで使用された 3,000 万のワークフローデータを大規模に調査し、チームが実際に、どのようにコミットからリリースまでを行っているのかを確認しました。次のいくつかの重要な質問への答えを得ることを目的に CircleCI はこの調査を実施しました。

- 「高速」とは実際にどのようなものか？
- さまざまなチームのデリバリーは、どのように異なっているか？
- データから得られたパフォーマンス、調査で報告されたパフォーマンス、およびベンチマークに関する業界の前提は、整合性があるのか、それとも異なっているのか？

最初に、いくつかの重要な用語について説明します。継続的インテグレーションと継続的デリバリー（CI/CD）は、ソフトウェアデリバリーの基盤です。CI/CD は、ソフトウェアのビルド、テスト、およびデプロイのために、バックグラウンドで実施される自動化された手順を定義します。これは、アジャイル開発を具体化した例です。DevOps は、絶え間ない変化が必然となっている、現代のソフトウェア開発のためのチーム構造、ツール活用、および組織文化に関わっています。CI/CD は、手動による多くの承認手順がこれまで必要であったプロセス（場合によっては数か月を要していた）を、数分で自動的に処理するための機能を提供します。

4 つの主要な統計である、リードタイム、デプロイ頻度、平均修復時間（MTTR）、および変更による失敗率は、監視可能な CI/CD メトリクスに次のように明確にマッピングされます。

- 変更のリードタイム → ワークフローの期間
- デプロイ頻度 → ワークフローを開始する頻度
- MTTR → 状況がレッドからグリーンになるまでに要する時間
- 変更による失敗率 → ワークフローの失敗率

上記の関係において、ワークフローは、ジョブのコレクションと実行順序を定義する一連のルールとして使用されます。これらの 4 つの主要なメトリクスを最適化することで、継続的インテグレーションとデリバリーに着手していない組織に比べて、大きなアドバンテージを得ることができます。これを裏付ける明確なデータがあります。

優れたパフォーマンスを発揮している DevOps チームは、組織の別のチームと比較して、デプロイ頻度が 200 倍、MTTR が 24 倍高速、変更による失敗率が 3 分の 1、リードタイムは 2,555 倍 * になっています。

ここでは、スピードだけが目標ではないことに注目することが重要です。自動化、プロセスからの手動タスクと人的な遅延の排除、およびテストの前倒しを総合的に進めることで、スピードと品質が向上します。**デリバリーの一貫性は、目立たないかもしれませんが、自動化の大きな成果の 1 つです。信頼性や品質を向上させることのない、単なるスピードアップには意味がありません。**

これらの主要なメトリクスを管理できるようになるには、開発パイプラインで対応する CI/CD の要素を理解する必要があります。では、各要素を見てみましょう。

* *Puppet 2016 State of DevOps Report*

リードタイム

リードタイムとは、ワークフローが最初に開始されてから完了するまでにかかる実行時間の長さです。これは、シグナルを取得できる速さの測定基準となります。リードタイムを短縮するには、ソフトウェア開発パイプラインを最大限に自動化する必要があります。チームが、継続的インテグレーションを使用してビルドとテストを自動化するのは、この目的のためです。CI によるパイプラインの自動化により、数週間から数か月かかっていたデプロイタイムラインが、数分までとはいかないまでも数時間に短縮されます。CI を使用するチームは、CI を使用していないチームと比較して、リードタイムを 4 桁も短縮できます。

ワークフローには多くの種類があります。別のワークフローによって後でピックアップされるホスティングサービスにアーティファクトをデプロイするワークフローもあるかもしれません。したがって、このリードタイムにおけるワークフローとは、「本番稼働環境にデプロイする」ことに限らず、ワークフローの目的となる最終的な状態、つまりワークフローの成果に到達することを意味します。測定されるリードタイムは、これらのワークフローの実行にかかる時間に基づいています。

ワークフローが実行される時間の長さにより、CI/CD の中核であるフィードバックサイクルが定義されます。優れたパフォーマンスを発揮しているチームには、エラーを発生させずにコードを生成できることや、すべてのコード変更におけるグリーンビルドが求められているわけではありません。レッドビルドを恐れる必要はありません。これらは、チームのスキルレベルに関わらず、開発プロセスで必ず発生するものです。チームは、障害を恐れるのではなく、障害から可能な限り多くの情報を取得して、障害にできるだけすばやく対処できるようにする必要があります。チームは、何もせずに、ビルドを成功させるのではなく（グリーンビルド）、問題解決のプロセスを経て、グリーンビルドを実現できるようにする必要があります。テストを行わなければ、ワークフローはすばやく実行され、必ず成功します。しかし、これは何も役立ちません。テストを行わなければ、ワークフローは高速に実行され成功し、コードが求められている基準を達成できるかどうかについて、エンジニアに有益なシグナルが提供されることもありません。

極めて高速なワークフローが役立たないとしたら、長時間を要するワークフローのほうが良いと言えるでしょうか？チームが目標とするべきワークフローの長さはあるのでしょうか？ワークフローに数十時間かかる場合、複雑で包括的なテストスイートであることを示している場合があります。その実行時間はワークフローが達成する目標を評価するためにある程度効率的に活用できるかもしれませんが、有用なシグナルを得るために、チームはあまりにも長い時間を待機することになります。**最適なワークフローは、数分から数 10 分程度で実行されます。**これにより、1 日を通してコード変更をプッシュし、失敗したワークフローに迅速に修正を適用できる十分な頻度で、開発者にシグナルを送信できます。

デプロイの頻度

チームがワークフローを開始する頻度は、開発パイプライン内を移動している個別の作業単位の数を示します。ソフトウェア開発パイプラインを完全に自動化すると、コードベースに対する複雑なアップデートであっても、自動的にデプロイできます。修正プログラムはすぐに利用可能になり、他のアップデートと同じように、厳格なテストを適用できます。このメトリクスの重要な点は、いつでも自由にデプロイできるチームに関する能力の指標として利用できることです。

CI パイプラインの信頼性、つまり、デプロイを自動化する機能により、コードを手動で検証する労力を排除でき、デプロイの頻度を向上することが可能になります。テストはすべてのコミットで実行され、失敗したビルドが通知され、失敗を瞬時に確認できます。エラー情報をできるだけ早く入手できれば、変更をさらに迅速にプッシュでき、デプロイ頻度がさらに向上します。

平均復旧時間 (MTTR)

平均復旧時間は、実用的な情報を迅速に取得できるかどうかによって大きく左右されます。失敗のステータスを迅速に確認できれば、最短でレッドからグリーンに移行できます。CI/CD の手法を採用すると、迅速なフィードバックループが可能になり、開発者にすばやくシグナルを送信できます。堅牢な CI/CD の手法には、テストスイートのログやカバレッジレポートなどのリアルタイムアーティファクトが含まれ、開発者は本番稼働環境と同等の環境でトラブルシューティングを行うことができます。

変更による失敗率

優れたチームであれば、不正なコードをデフォルトブランチにプッシュすることはまずありません。しかし、このようなチームが、不正なコードを決して書かないという意味ではありません。テストとセキュリティチェックは別のブランチで実行され、すべてが合格したときに、マージの実行が許可されます。チームは、デフォルトブランチにマージされる前に、コードが適切に機能することを把握しておく必要があります。トピックブランチは、シグナルを最速で取得する場所であり、失敗しても安全な場所です。このプラクティスは、Vincent Driessen 氏が提唱する Git-flow モデルに倣った開発戦略です。トピックブランチにおける変更による失敗率はデフォルトブランチにおける失敗率よりも高くなります。トピックブランチでは作業の大部分が実行され、これらのブランチの失敗が、デフォルトブランチに降りてくることはないので、トピックブランチで発生する障害は、コードベース / 製品全体ではなく、同じブランチで作業しているユーザーにのみ影響します。

チームメンバーがデフォルトブランチで直接開発する一般的な別の開発戦略である「トランクベース開発」を行っている他のチームについても CircleCI はデータを収集しました。この戦略では、ビルドがレッドの場合、すべてのユーザーが復旧に取り組むことから、復旧時間が最適化されます。しかし、この方法が使用されたデータについてはほぼ収集できませんでした。



これらのデータが意味することとは？

次のセクションでは、2019 年 6 月 1 日から 8 月 30 日までに確認された 3000 万を超えるワークフローデータについて詳しく見ていきましょう。

このワークフローデータは以下を示しています。


- 1 日あたり 160 万回のジョブ実行
- 40,000 以上の組織
- 150,000 以上のプロジェクト

リードタイム

リードタイムは、ワークフローが最初に開始されてから完了するまでの実行時間の長さです。CircleCI のデータセットで記録された最小リードタイムは 2.1 秒でした。2.1 秒で完了するワークフローとは何を実行するものでしょうか？あまり多くの処理はできないはずです。リードタイムが 2.1 秒になるシナリオには、通知を送信する、あるいはメッセージを出力して終了コード 0 を返すこと処理などが考えられます。

最大リードタイムは 3.3 日でした。このシナリオは、複合的なテスト、複合的なリグレッションテスト、複合的なインテグレーションテスト、LDAP ディレクトリにあるデータベースの外部依存関係のスピンアップ、パフォーマンステストの実行、また複数のプラットフォーム向けのクロスコンパイルなど、大量のテストである場合があります。

調査した 3,000 万のワークフローの 80% が 10 分以内に終了していました。リードタイムの 50 パーセンタイルは 3 分 27 秒でした。95 パーセンタイルは 28 分でした。シグナルを受け取るまでに 28 分間待機するのは長すぎると思うチームもあるでしょう。また、ワークフローを 28 分で実行できれば大成功とするチームもあるでしょう。ワークフローのリードタイムは、ビルドしているソフトウェアの種類、テストの統合レベル、テストの複雑さ、テストの程度に大きく依存します。CircleCI では、上記の要素が一定ではないため、ワークフローの理想的な長さについて普遍的な標準があるとは考えていません。普遍的なベンチマークを達成することを目指すのではなく、ワークフローの長さを短縮できる機会を探すべきだと CircleCI は考えます。**ワークフローの時間短縮につながる最適化は、歓迎すべきことです。**



3 日以上のワークフローがある組織を一律に遅いと判断するべきではありません。組織には、複数のプロジェクト、場合によっては数百のプロジェクトがある場合もあり、プロジェクト間の依存関係の構造はこのデータには反映されていないからです。リードタイムが数分から十数分の間で大きく異なる場合があるのにはさまざまな理由がありますが、重要な点は、CI/CD を使用していないチームと比較すると、CI/CD を使用しているチームは、リードタイムが大幅に短縮していることです。CI/CD は、セキュリティやコンプライアンスなど、これまで手動で実施されていた多くの承認を自動化し、承認やその他の手動ゲートの待ち時間を排除し、リードタイムを大幅に削減します。

デプロイの頻度

2009 年、Flickr の John Allspaw 氏と Paul Hammond 氏が講演を行い、自社のチームが 1 日に 10 回デプロイしていることを説明しました。それから、多くのチームがこのデプロイ回数を超えるように懸命な努力を重ねており、多くのチームが非常に高い頻度でデプロイしているはずだと思っていましたが、多くのチームが 1 日に数十回以上展開しているというデータは確認できていません。このような運用も存在しますが、例外であり、ルールではありません。これだけ大きな格差がある理由は、デプロイの頻度が、チームが開発しているソフトウェアに大きく依存するためです。

CircleCI を利用している 50% の組織は、すべてのプロジェクトで 1 日に 6 件のワークフローを開始しています。プロジェクトレベルでは、プロジェクトの半数は 1 日に 3 つのワークフローを開始し、半数はデフォルトブランチで 1 日に 1 件のワークフローのみを開始しています。そして、最上位レベルの 95 パーセントでは、これらの数値は、デフォルトブランチで 39 件のワークフロー、プロジェクトあたり 74 件のワークフロー、組織のすべてのプロジェクトで 250 件のワークフローに増加します。

CircleCI を使用している組織は、必要なときにいつでもデプロイできるため、デプロイ回数は、組織の事情を反映したものになり、必ずしも優れた開発チームかどうかの指標にはなりません。代わりに、組織がコードをテストおよび統合する頻度と、エラーが発生したときにどれだけ早く復旧できるかという指標は、開発チームのパフォーマンスが高いことを示します。

平均復旧時間

CircleCI で記録された復旧時間の最小値、1 秒未満です。これが起こり得るのは、2 つのワークフローをほぼ同時に開始し、一方が成功し、他方が失敗する場合です。他のシナリオでは、これだけの短時間にシグナルを取得してそのシグナルに応答することはできません。

記録された最大の復旧時間は 30 日でした。これは、恐らく、復旧が 30 日間行われているのではなく、CircleCI のデータセットでは、30 日間で復旧期間が途切れるために、この最大値になったのでしょう。

CircleCI における復旧時間の中央値は 17.5 時間です。これは、ある就業日が終了してから次の就業日が開始するまでの時間の長さです。これは、エンジニアが 1 日の終わりに失敗のシグナルを受け取った場合、問題を解決するために翌日まで待機していることになります。

ここで興味深いのは、このデータは、最小時間でも最大時間でもなく、中央値の時間であることです。この 17.5 時間の中央値は、平均復旧時間はこの数値よりもはるかに短いとする調査結果を示す一般常識とはかけ離れています。

では、報告されている一般のデータと CircleCI で確認されたデータ間で差異がある要因は何でしょうか？ CircleCI は、調査の設計自体が報告される MTTR に影響している可能性があると考えています。調査の質問は、多くの場合、「自社で取り組んでいる主要なアプリケーションまたはサービスは何ですか？」というものです。CircleCI のデータは、デフォルトブランチだけでなく、プライマリアプリケーションだけでもなく、すべてのブランチのすべてのワークフローを対象としています。確認された復旧時間の中央値は 17.5 時間ですが、プライマリアプリケーションのデフォルトブランチでの平均復旧時間は、数分から数時間であり、この数値よりもはるかに短くなっています。CircleCI のデータによると、アクティブなプロジェクトブランチの 30% 以上は、調査された 90 日間で失敗することがありませんでした（100% グリーン）。つまり、失敗することがなかったため、復旧の必要もないため、MTTR は 0 になります。

CircleCI の MTTR に関する注意すべき他のデータは、**すべての復旧の 50% が 1 時間以内に行われており、25% の組織は 15 分以内に復旧していることです**。組織の 50% は 1 回の試行で復旧し、75% は 2 回の試行で復旧しています。上位 10% のパフォーマンスにランクインするチームは、ビルドを修正して正常な状態（グリーン）に戻すために必要な作業を 10 分未満で完了しています。また、興味深いのは、3 時間と中央値 17.5 時間というデータには頻度に大きなギャップがあり、レッドビルドが 3 時間未満で復旧しないと、翌日まで復旧しない可能性が高いことを示唆しています。

変更による失敗率

全体でみると、CircleCI のすべてのワークフローの 27% が失敗しています。各ブランチは常にグリーンである必要があると考えている人もいますが、レッドビルドであっても、迅速に復旧できる限り問題はありません。ビルドの失敗は、決して悪い兆候ではありません。CI システムが機能しており、有効なデータを提供していることを意味します。ブランチタイプ別の障害を確認し、何が起きているのかをさらに詳しく見ていきましょう。トピックブランチの平均障害率は 31% です。しかし、デフォルトブランチに限定すると、変更による失敗率は 18% に低下します。デフォルトブランチの失敗率が低いことは驚くことではありません。前に説明したように、アクティブなプロジェクトブランチの 30% 以上が失敗することはありません。この平均は、トピックブランチはメインラインに統合される前に変更の大部分がコミットおよび検証される場所になっているという、CircleCI の仮定を裏付けています。トピックブランチで広範な事前作業を行った後に、デフォルトブランチにマージすると、コードによって生じる変更をより正確に把握できます。

CI/CD のプロセスをオーケストレーションする circle.yml ファイルの構成が変更された場合、プロジェクトの 50% で障害が発生しなかったことは、驚きでした。障害発生率がこれほど低い理由の 1 つは、構成が再利用されることでしょう。たとえば、同じ構成の多くのプロジェクトを実施している組織は、変更がテストに合格すると、プロジェクトを更新して複製しています。別の理由は、CircleCI Orbs の使用です。Orbs は再利用可能で共有可能な構成パッケージです。Orbs は作成者によってテストされ、コミュニティによって検証されているため、失敗を恐れることなく、機能を追加するために使用できます。どのような理由で障害率がこれほど低くなっているかに関わらず、構成変更は難しく、頻繁な更新が必要となるという広く認識されている常識に反していることは、CircleCI のユーザーにとって重要なことです。

このデータの意味するものは？

このデータは、CI/CD ツールの一環として CircleCI を使用しているチームが、最高のパフォーマンスを発揮していることを示しています。

ソフトウェアデリバリー パフォーマンスの要素	エリート *	平均的な CircleCI ユーザー
デプロイ頻度 → ワークフローの開始頻度	オンデマンド： 1 日に複数回デプロイ	1 日に 6 件のワークフロー
変更のリードタイム → ワークフローの期間	1 日未満	3.5 分
平均復旧時間	1 時間未満	1 時間未満
変更による失敗率 → ワークフローの失敗率	0 ~ 15%	デフォルトブランチで 18%

* 『2019 Accelerate State of DevOps Report』を出典とするエリートメトリクス。

しかし、このデータは CI/CD ツールを採用しているすべてのチームのデータを反映しているのでしょうか？また、上記の図の DevOps の原則を採用し、CircleCI を使用して導入しているチームをより正確に反映しているのでしょうか？

CircleCI が提供している機能を使用すると、DevOps の手法を取り入れた最も効率的なチームになります。たとえば、並列処理とインテリジェントなテスト分割により、CircleCI でビルドを同時に実行し、リードタイムを劇的に短縮できます。Docker レイヤーキャッシングや他の依存関係キャッシングなど、CircleCI の他の高度な機能により、大規模な依存関係やアーティファクトを複数回ロードするのではなく、一度にロードでき、ビルドの高速化とリードタイムの短縮を実現できます。

他の CircleCI 機能を使用することで、リードタイムの短縮と MTTR 改善をさらに推進できます。さまざまな多くのリソースクラスでビルドするオプションを利用できるため、CPU、GPU、および RAM リソースを構成して、リードタイムを短縮できます。また、チームはビルドが失敗した特定のマシンに SSH で接続できるため、問題が発生した正確な環境でレッドビルドを簡単にトラブルシューティングして、復旧時間を短縮できます。最後に、ログとテスト結果に簡単にアクセスできるダッシュボードを使用できるため、CircleCI でビルドしているチームの復旧時間をさらに短縮できます。

人気のある CircleCI の機能である Orbs を使用すると、コミュニティおよびパートナーが開発した、すぐに使用できる安定しており信頼して使用できるソリューションで、コンフィグを更新できます。Orbs を使用すると、変更による失敗率を低減し、コンフィグファイルを変更するときのパイプラインの信頼性と一貫性を向上できます。これは CircleCI ユーザーにとって大きな利点です。これらの再利用可能なコードスニペットは、繰り返しのプロセスを自動化して、プロジェクトの設定を高速化し、サードパーティアプリケーションとの統合を容易にします。



デリバリー成果の向上は、CI から始まる

これらの DevOps の効率性を示すメトリクスは重要であり、エンジニアリングチームの成果を向上することは広く理解されていますが、この課題を達成する方法はまだ解決されていません。しかし、エンジニアリングの成果を向上したいと考えているチームにとって、CircleCI のデータは非常に役立つものです。CI/CD の原則を採用する利点を享受するためには、専門家である必要も、すべてを完璧に実施する必要もありません。実際、CircleCI の平均的なユーザー（50 パーセントのユーザー）は、CI/CD の原則を採用するだけで、高いパフォーマンスを達成しています。

しかし、1 つのツールや手法を導入しても、チームの成功にとって特効薬にはなりません。新しい手法を導入すると、必然的に失敗率は増加しますので、組織全体で変化を取り入れることに恐怖を感じるかもしれません。しかし、CI/CD を組織に導入することは、この文化を変える触媒となる可能性があります。このレポートでは、CI/CD の手法が成功とどのように関係しているのかを示しました。

これらの変化を実現することは簡単ではありません。優れたソフトウェアを作り出すことも容易ではなく、CI/CD ツールの採用は一晩で完了するものでもありません。しかし、これらの DevOps の手法を改善することは、最高のチームを作り出し、一貫して、自信を持って、迅速に最高のソフトウェアを作成する方法です。優れたパフォーマンスを発揮しているチームを目指すときに、必ずしも CircleCI を使用する必要があるわけではありません。しかし、CircleCI のプラットフォームで CI をホスティングするチームは、その分野のリーダーとなっていることも事実です。

今後の調査

今回の分析により、今後調査すべき追加領域が明らかになりました。

- 言語固有の差異がいくつか見つかり、今後の調査で検証が必要です。たとえば、PHP で書かれたプロジェクトのデフォルトブランチの失敗率は、すべてのデフォルトブランチの平均失敗率の 18% と比較して、わずか 7% でした。PHP プロジェクトで実行されているテスト数が少ない(またはテストがまったく実行されていない)のか？ PHP の専門家であるエンジニアが多数存在しているのか？その理由は不明です。JavaScript、Python、Ruby、Go、および Java についても興味深いパターンが検出されています。これらのパターンについては、今後詳しく調査する予定です。
- また、ブランチ数が成功率と速度に影響を与えるかどうか、およびチームの規模に応じてブランチの明確なパターンがあるかどうかについてさらに調査する予定です。
- 最後に、CircleCI のユーザーが、コンフィグで手動の承認手順をどのように使用しているのか、そしてその方法が、チームのリードタイムと変更による失敗率にどのように影響しているのかを調査する予定です。

調査の手法について

CircleCI のこのデータは、2019 年 6 月 1 日から 8 月 30 日までの 90 日間で収集されました。3100 万のデータポイント、40,000 以上の組織、および 150,000 以上のプロジェクトがこのデータに含まれています。