Engineer

DevOps

∞ ソフトウェアの開発と運用を結び付ける実践

どのようなものか

DevOpsは、システムを開発する人が、ビルド、リリース、稼働の維持にも責任を持つ、ソフトウェア開発の進め方です。コードを書いてから動かすまでの手順を、チーム間で引き渡すのではなく、自動化します。名前はdevelopmentとoperations、開発と運用を組み合わせたもので、この二つの集団を結び付けるために作られました。

米国国立標準技術研究所、NISTは、開発チームと運用チームの間の処理を自動化し、「ソフトウェアを、より速く、より確実にビルド、テスト、リリースできるようにする」実践と定義します。有用な定義なのは、重要な部分、つまり両者の間の処理を明示しているためです。開発者と運用担当者がいずれも優秀でも、一方が他方へリリースを引き渡す場所で、何日も失う場合があります。

実務では、互いを補強する、いくつかの習慣として現れます。

- **本番の責任を共有します。** コードを変える人が、監視、ログ、アラートを通じ、稼働後に変更が何をしたかを見て、責任を負います。
- **リリースまでの経路を自動化します。** ビルド、テスト、デプロイを、毎回同じパイプラインで動かします。通常は、CIを土台にします。
- **インフラをファイルで記述します。** サーバー、ネットワーク、データベースを、コードと並べて保持するテンプレートやプレイブックで表します。環境を記憶するのではなく、再構築できるようにします。
- **一度のビルドを、多数の環境で使います。** 検証環境でテストを通った成果物と、本番へ届く成果物は同じで、変えるのは設定だけです。
- **小さく頻繁にリリースします。** 一回の変更が少ないため、不具合を見つけやすく、ロールバックもしやすくなります。

DevOpsは製品ではなく、職種名としても、大まかな呼び方にすぎません。売り手は、この名前で道具を売り、企業は「DevOpsエンジニア」を採用します。しかし、どちらも実践そのものではありません。DevOpsエンジニアは通常、ほかの開発者が自力でソフトウェアを届けるためのパイプラインとインフラを作り、維持します。実践の中の一つの役割であり、実践の代わりではありません。

概念を判断に生かす

「DevOpsとは？」を実際の判断に生かしましょう。その概念を、開発または購入するシステムに結び付けて考えるお手伝いをします。 変更を検証し、リリースし、元に戻せる仕組みを整え、金曜の深夜ではなく火曜の午後にお客様へ届けられるようにします。

サービスを見る お問い合わせ →

用語の背後にある前提を検証する

解説とリンク先の実績を参考に、プロジェクトについての質問を整理してください。ツールを選ぶ前に、前提を一緒に検討できます。 GitHub ActionsでGIS SaaSのデプロイを自動化 — デプロイ、データ、報告を手作業から解放

GitHub ActionsでGIS SaaSのデプロイを自動化 お問い合わせ →

なぜ重要なのか

チーム間で引き渡すたびに、何かが失われます。開発者は、なぜ変更し、何に依存するかを知っています。運用担当者は、サーバーの構成と、先月何が壊れたかを知っています。リリースが、チケットとパッケージの形で両者の間を渡ると、それぞれの知識が、不完全な形でしか届きません。欠けた部分が表に出るのは、最も負担の大きい、デプロイの最中です。

SaaS（サービスとして提供するソフトウェア）アプリの構築方法を示すTwelve-Factor Appは、三つの隔たりを明確に挙げます。今日書いたコードが何週間も後に本番へ届く、時間の隔たりがあります。書く集団とデプロイする集団が違う、人の隔たりがあります。ノートPCのデータベースと本番のデータベースが違う、道具の隔たりがあります。DevOpsの多くは、この三つを埋める取り組みです。書いた数時間後にデプロイし、書いた本人がデプロイし、どの環境も、できるだけ似たものに保ちます。

第二の理由は、リリースの大きさです。Googleのサイト信頼性エンジニアリングの本は、頻繁なリリースなら、版の間の変更が少ないと、明確に述べています。三つの変更を含むリリースなら、失敗したときに、何が起きたかを考えやすくなります。三百の変更なら、原因を探し回る仕事になります。

第三は、トイルです。同じ本は、サービスの運用に伴い、手作業で、繰り返しがあり、自動化でき、持続的な価値を生まず、サービスとともに増える仕事と定義します。毎週火曜日に手でプロセスを再起動し、カレンダーの通知で証明書を更

新し、アラートが鳴るとディスクのサイズを変更するチームは、スクリプトの方が確実に行える仕事へ時間を使っています。自動化することで、小さなチームでも、成長するシステムに対応できます。

セキュリティーの指針も、同じ構成を採っています。クラウドネイティブアプリのDevSecOpsに関するNISTの指針は、ビルド、テスト、パッケージ化、デプロイ、運用という段階を持つパイプラインから始めます。そのパイプラインが記述するコードの種類に、インフラ、実行時のポリシー、監視も含めます。

実際の活用例

このサイトのページは、DevOpsを職種名よりも、責任範囲として示しています。同じエンジニアが、システムを設計し、本番へ届け、その後も稼働を維持します。

ある地図製品では、創業者の仕事が、インフラ、データ処理、顧客向けの地図アプリに、二年間にわたって及びました。各層のアーキテクチャ、実装、サポートを担当し、動作環境、データ処理のワークフロー、地図アプリを、設計し、構築し、維持しました。一つの仕事で、開発と運用をつないでいます。記録が示すのは、このエンジニアリングと保守の範囲と期間であり、稼働率や応答時間の測定値ではありません。

Ubuntu LinuxのVPS上にある、PostgreSQLとMS SQL Serverの10インスタンスでは、運用を少しずつ自動化しました。`pg_dump`とSQL Server Agentジョブによるバックアップ、PrometheusとGrafanaによる監視、定期的なパッチ適用、UFWファイアウォール、ロールベースのアクセス制御、文書化した災害復旧手順を整備しました。ページは、十台すべてを対象に99.9%の稼働率を報告し、自動化で手作業の保守を50%減らしたとしています。一つのサーバー群で測った、トイル削減の例です。

開発、検証、本番の環境を持つAzureプラットフォームでは、固定した経路に沿って、デプロイ全体に責任を持ちました。その経路の、すべての部分を担当するという意味です。CI/CDがイメージを作り、Bicepテンプレートが各段階のインフラを対応する構成に保ち、段階ごとの設定は、シークレットと分けて保存します。一度ビルドした成果物を、各段階で作り直さず、次へ進めます。変更は、場当たり的な手動デプロイを一度も挟まず、予測できる形で、各環境へ届きます。

問題が起きるところ

DevOpsは、静かに失敗します。たいていは、言葉だけを残し、習慣を失う形です。

職種名は変わっても、壁は残ります。 運用チームを改名したり、開発者と本番環境の間に置く「DevOpsチーム」を採用したりします。リリースの引き渡しは残り、相手の名前だけが変わります。確かめ方は簡単です。誰が変更を届けられ、壊れたら誰が起こされるかを尋ねます。違う情報を持つ別々の人なら、実践は、まだ実現していません。

デプロイは、今も人がチェックリストをたどる作業です。 パイプラインがビルドとテストを終えると、誰かがサーバーへログインし、最後の五段階を、記憶に頼って実行します。毎回、少しずつ違います。最後の段階もパイプラインへ入れ、環境間を進む経路を一つにします。本番への手動プッシュは、説明すべきインシデントとして扱います。

環境ごとに、別のビルドを作ります。 検証用と本番用に作り直すと、試したものと届けたものが別になります。二回目のビルドで、依存関係の解決結果が変わると、誰も再現できない不具合として現れます。一度ビルドし、元のコミットで成果物にタグを付け、その成果物を次の環境へ進めます。設定は、環境から受け取ります。

```
tag="$(git rev-parse --short HEAD)"
docker build -t "registry.example.com/app:$tag" .
docker push "registry.example.com/app:$tag"
# 検証環境と本番環境は同じタグを使う
# 起動時に渡す設定だけが異なる
docker run --env-file staging.env "registry.example.com/app:$tag"
```

検証環境と本番環境が、離れていきます。 一台だけ手で変えた設定、本番から先に更新したデータベース、一方では固定し、他方では固定しないライブラリーの版は、どれも、予行演習としての検証環境の価値を下げます。インフラをテンプレートに記述し、どこでも同じテンプレートを適用します。開発でも、本番と同じバックエンドサービスを動かします。

システムとともに、トイルが増えます。 一台なら週に10分で済んだ手作業が、二十台では何時間もかかります。意識して自動化の時間を確保しなければ、チームは、稼働を保つだけで日々を使い切ります。繰り返す手作業を数え、使ってよ

い時間の割合に上限を設けます。

参考資料

- DevOps — NISTの定義と、出典である特別刊行物です。
- ビルド、リリース、実行 — 一度だけビルドし、設定と組み合わせてリリースを作る理由です。
- 開発と本番の同等性 — 開発と本番の間の時間、人、道具の隔たりと、埋め方です。
- 設定 — デプロイごとに変わるものを、コードの外、環境の中へ置く考え方です。
- トイルの除去 — Googleによるトイルの定義と、上限を設ける理由です。
- リリースエンジニアリング — ハーメチックビルド、セルフサービスのリリース、リリース用の道具で強制するポリシーです。

用語集

実際の制約に合わせて選ぶ

デプロイの流れ、運用上のリスク、緊急対応の担当者をお知らせください。検査、復旧手段、明確な責任の所在を伴う変更の範囲を定められます。

料金を見る お問い合わせ →

用語を実務に結び付ける

「DevOpsとは？」がお客様の組織にどう当てはまるかをご相談ください。現在のシステムの具体例が一つあれば、話を始められます。

サービスを見る

welcome@engineer.company +45 50 10 61 84 デンマーク、コペンハーゲン

このページを持ち歩く

連絡先カード

心を込めて丁寧に制作： D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

著作権 © 2025 – 現在 · Engineer ApS · すべての権利を留保します。

印刷元： https://engineer.company/ja/glossary/devops/