Engineer

デプロイ

変更を実際に利用される場所へ届けること

どのようなものか

デプロイは、アプリケーションのテスト済みバージョンを、ビルドシステムから取り出し、実際のサービスへ移す一連の手順です。ビルドしたファイルやコンテナーイメージを、実行するマシンやプラットフォームへ配置し、その環境の設定を渡し、起動し、実際のリクエストをそちらへ切り替えます。作業と、その結果の両方を指すので、チームは、データベースの段階で止まったデプロイとも、各環境で現在動いているデプロイとも言います。対象は、ウェブサイト、API、定期ジョブ、データベースの変更、モバイルアプリの背後のバックエンドなどです。

デプロイとリリースは、分けることがよくあります。デプロイは、新しいコードを実行できる場所へ置くことです。リリースは、誰に見せるかを決めることです。フラグで隠した機能を月曜日にデプロイし、木曜日に顧客へ公開することも、サーバーに再度触れずにできます。分離すれば、コードを小さな変更に分けて頻繁に届けながら、公開時期に関する製品側の判断は、その責任者に任せられます。

現在のデプロイの多くは、コンテナーイメージを運びます。2015年6月にLinux Foundationの下で設立されたOpen Container Initiativeは、関連する三つの仕様を公開しています。イメージの構成、展開済みイメージをランタイムが実行する方法、レジストリーによる保存と配布です。イメージは、内容のハッシュであるダイジェストで識別するため、同じダイジェストは、常に同じバイト列を表します。`1.4.2` のようなタグは、人が読める形でイメージを指す参照です。同じタグで再度プッシュすると、別のイメージを指すようになります。

ウェブサービスの構築方法として広く引用されるTwelve-Factor Appは、コードから稼働するサービスまでを、厳密に分けた三段階にします。ビルド段階では、コミットを実行可能な成果物一式へ変換します。リリース段階では、それと一つの環境の設定を組み合わせます。実行段階では、選んだリリースからプロセスを

起動します。各リリースには一意の識別子を付け、作った後は変更しません。変更すれば新しいリリースになるため、ロールバックは、以前のものを選ぶ作業になります。

切り替え方は、サービスがどこまで許容できるかで変わります。再作成型のデプロイは、古い版を止めてから新しい版を起動し、間の停止を受け入れます。ローリング型は、インスタンスを少数ずつ置き換え、一定期間、両方の版がトラフィックを処理します。KubernetesのDeploymentでは、これが既定です。ブルーグリーン型は、新しい版を古い版の隣で動かし、全トラフィックを一度に切り替えます。古い版は、トラフィックを戻せる状態で維持します。カナリア型は、まず新しい版へ送るトラフィックの割合を小さくし、エラー率とレイテンシーが健全な間、その割合を増やします。

概念を判断に生かす

「デプロイとは？」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

デプロイは、変更が誰かの役に立ち始める時点であり、間違いが利用者へ届き始める時点でもあります。めったにデプロイしないチームは、一回のリリースに多くの変更をまとめがちです。そのため、一回のリスクも、失敗時の原因候補も増えます。小さく頻繁にデプロイすれば、各段階を理解しやすく、戻しやすく保てます。

手作業のデプロイがどれほど一貫するかは、実行する人次第です。手順を抜かす、順番を変える、休暇中の担当者を代行する同僚が少し違う方法で行う、といったことが起きます。パイプラインは、毎回同じ手順を実行し、コミット、成果物、環境、時刻を記録します。インシデントの振り返りは、そこから始めます。この記録を見れば、何が変わったかを最も早く確認できることもよくあります。

最も確かな習慣は、一度だけビルドし、その結果を次の環境へ進めることです。検証環境でテストした成果物は、本番へ届くものと、バイト単位で同一であるべきです。違うのは設定だけで、成果物が届いたときに、その環境が設定を渡しま

す。ビルド時にダイジェストを記録し、それを指定してデプロイすれば、この約束を検査できます。

```
podman build -t registry.example.com/web:1.4.2 .
podman push --digestfile web.digest registry.example.com/web:1.4.2
kubectl -n staging set image deployment/web \
  web=registry.example.com/web@$(cat web.digest)
# 検証環境のチェックに通ったら、同じダイジェストを本番へ進める
kubectl -n production set image deployment/web \
  web=registry.example.com/web@$(cat web.digest)
```

ロールバックも、同じ規律に依存します。リリースが不変で識別されていれば、前のものを指すだけで、数秒で戻れます。各環境がソースから作り直す方式では、昔のコミットを再ビルドし、依存関係が以前と同じように解決することを期待する作業になります。

実際の活用例

あるプラットフォームのAzureへのデプロイを、最初から最後まで担当する仕事では、すべての変更が、開発、検証、本番という同じ固定経路を通り、稼働中の環境へ直接プッシュする人はいませんでした。CI/CDがイメージを作って運び、Bicepテンプレートが各環境の対象インフラを同じに保つため、次の段階へ進めるたびに、対応するインフラへ届きます。環境ごとに違う設定は、シークレットと分けて保持します。そのため、一度ビルドした成果物が全段階を通り、到着時に各環境の設定を受け取れます。途中で作り直しません。変更は、定義した経路で各環境へ届き、場当たり的な手動デプロイはありません。リリースは、毎回同じように実行する、管理された段階になりました。

以前の仕事は、一貫性の価値を時間で示しています。ビルド、テスト、デプロイの自動化で、約4,000時間を節約しました。変更から本番までの経路を標準化し、実行者によって異なっていた手作業をなくした成果です。リリースは、速くなり、信頼性も上がりました。

自社のプラットフォームでは、デプロイとプロビジョニングを分けています。PodmanとQuadletによるコンテナー層では、rootが一度だけデプロイ経路を準備します。その後は、非特権ユーザーが、固定した一つのスクリプトを実行して再デプロイします。範囲を絞った規則によって、引数なしの、その一つのコマンドだけを許可します。スクリプトがイメージをビルドし、サービスを再起動するため、通常のリリースにrootのシェルも、任意のコマンドも必要ありません。

問題が起きるところ

デプロイの問題の多くは、確かめたものと、実際に変わったものの差から生じます。

検証環境で見ていないものが、本番で動きます。 環境ごとに作り直すと、依存関係に指定したバージョン範囲から新しい版が選ばれる、ビルド間にベースイメージが変わる、ビルドのフラグが違う、といったことが起きます。テストを通った成果物と、顧客に応答する成果物が別になります。一度ビルドし、ダイジェストを記録し、同じダイジェストを指定して次の環境へデプロイします。環境固有のものは、成果物に入れず、設定に保ちます。

タグが動き、ロールバック先を間違えます。 `latest` を指定するデプロイや、再ビルド後にバージョンタグを再利用するデプロイは、その日の参照先次第になります。Kubernetesの文書は、本番の `:latest` を勧めません。稼働中の版を特定しにくく、正しくロールバックしにくくなるためです。再利用しないバージョンタグを使い、対応するプラットフォームでは、ダイジェストを固定します。

新しい版が動く前に、パイプラインが成功を報告します。 多くのデプロイコマンドは、プラットフォームが要求を受け付けると、すぐ戻ります。新しいインスタンスはまだ起動中で、一分後にクラッシュする可能性もあります。新しい版が準備完了を報告するまで待ち、実際のリクエストをいくつか送って応答を検査すれば、静かな故障を、ビルドの失敗として表面化できます。前のリリースへ、まだすぐ戻れるうちに分かります。

展開の途中で、新コードと旧スキーマが出会います。 ローリング型やカナリア型では、新旧の版が並行して、一つのデータベースに対してトラフィックを処理します。列名を変える、列を消すといった移行は、稼働中の古いインスタンスを壊します。古いコードは、消したものを期待しているため、ロールバックも危険になります。変更を複数のデプロイに分けます。先に新しい構造を追加し、両方で動くコードを届け、稼働中のどの版も必要としなくなってから、古い構造を消します。

参考資料

- ビルド、リリース、実行 — 三つの段階を分け、作成後は変更しないリリースを使う、Twelve-Factor Appの考え方です。
- Open Container Initiative — 多くのコンテナーデプロイが使う、イメージ、ランタイム、配布の仕様を担うLinux Foundationのプロジェクトです。

- OCIイメージ形式仕様 — マニフェスト、レイヤー、設定、イメージを識別する内容のダイジェストです。
- KubernetesのDeployment — ローリング更新、再作成の戦略、進捗の期限、ロールバックを説明します。
- Kubernetesのイメージ — タグ、ダイジェスト、取得ポリシーと、`:latest` がロールバックを難しくする理由です。

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

「デプロイとは？」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/deployment/