Engineer

継続的デリバリー

必要なときに安全にリリースできる状態を保つこと

どのようなものか

継続的デリバリーとは、必要なときにソフトウェアの変更を安全にリリースできるようにすることです。変更には繰り返し使える経路があり、各段階の結果から次に進めるか判断できます。本番に反映する時刻は、運用上の都合やメンテナンス時間、担当者の承認に合わせて決めてもかまいません。判断する時点で、検証済みのパッケージと、それを本番へ運ぶ手順が用意されています。

継続的インテグレーションでは、小さな変更を頻繁に統合して共有コードを確認します。ビルドできるか、既存のテストが通るかを早く把握できます。継続的デリバリーでは、作成した成果物を予定の環境に導入し、そこで動くことまで確かめます。継続的デプロイでは、条件を満たした変更が自動で本番に反映されます。リリース候補がすでにできていれば、人が本番のタイミングを決める場合でも継続的デリバリーは成立します。

コードを配置する時点と、利用者に機能を公開する時点も分けられます。新しいコードを本番に導入し、機能のスイッチを切ったまま動かす構成が一例です。その後、少人数から機能を有効にして範囲を広げられます。配置後は従来の処理が動くことを確認し、公開後は新しい機能を使う人に何が起きたかを観察します。誰がそれぞれの判断をするかも決めます。

概念を判断に生かす

「継続的デリバリーとは？」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

一人しか手順を覚えていない作業では、同じ変更でも結果が変わります。手順をワークフローにし、各段階で何を確認したかを書き残すと、次の担当者が理由を追えます。テストが失敗したのか、環境に接続できなかったのか、承認を待っているのかを区別できます。実際に導入したパッケージの識別子も記録します。`latest` のような移動するラベルだけでは、障害時に使われていた内容を調べにくくなります。

リリースが重い作業だと、別々の変更が次の機会まで積み上がります。障害が出れば、その一式から原因を探すことになります。繰り返せる経路があれば、小さく反映して一件ずつ影響を見られます。準備を終えたうえで、依存するサービスの保守が終わるまで待つこともできます。

一つのリポジトリだけで完結しない変更もあります。データベースの移行、ネットワークの規則、認証の設定には別の担当者がいるかもしれません。どの組み合わせが同時に動く必要があるかを整理し、関連する点で本番に近い環境を選んで確かめます。次に当番になった人にも、確認すべき設定と結果が分かる形で経路を残します。

実際の活用例

GitHub Actionsで環境間の昇格を含むCI/CDを構築した事例では、コードの確認、テスト、環境ごとのデプロイに別々のワークフローが使われています。フロントエンド、API、データベースの各イメージには、それぞれの用途に合うベースが選ばれました。この事例では、決められた経路と確認によってリリースが日常の作業になりました。別のプロジェクトでは、環境の構成や承認方法が変わります。

たとえば、アプリケーションにレポートの書き出しを追加します。ワークフローにソースのリビジョンを記録し、パッケージを一意に識別できるようにします。事前検証の環境で、権限のあるテストアカウントが実用に近いデータからレポートを取得します。列の内容とアクセス条件を確認し、承認する人は試験結果をパッケージの識別子と一緒に見ます。本番には同じパッケージを本番用の設定で配置します。配置後は短いテストで新しい書き出しと従来の操作を確認します。

検証で失敗したときは、パッケージと停止理由を残します。本番で失敗したときには、直前のパッケージと最初に悪化した指標が分かるようにします。検証環境で依存先を一つ意図的に停止し、どの段階で検出できるかを試せます。データベ

ースの変更を含むときは、中間状態のスキーマで以前のアプリケーションも動くかを確認します。

問題が起きるところ

環境ごとに作り直すと、試験を通った内容と本番に入る内容が異なる場合があります。同じコミットでも依存先やコンパイラが変わり得ます。互換性のある環境では、識別子を記録したパッケージをそのまま次に送ります。対象ごとに別のビルドが必要なら、成果物を個別に識別し、その対象に合う試験結果を付けます。

承認が別の手作業になってしまうこともあります。承認者が必要なのは、対象の成果物、環境、確認結果、残っているリスクです。その人が毎回すべてのコマンドを手でやり直すなら、最後の大事な部分は他の人に引き継げません。判断が必要な箇所を決め、それ以外は権限を持つ別の担当者でも同じように進められる形にします。

再実行のたびに結果が変わるテストでは、緑の印だけで判断しにくくなります。何が失敗を起こすのかを調べ、正常性を示す指標が実際の使い方を反映するかを確認します。影響の大きい変更では、対象を広げる前に観察する時間を設けます。停止する条件と担当者を決めておけば、中断後は把握済みのパッケージと検証した状態から再開できます。

参考資料

- DORAの継続的デリバリーは、必要時に公開できる能力を説明しています。
- DORAの継続的インテグレーションは、頻繁な統合と早いフィードバックを扱います。
- DORAのデプロイ自動化は、環境にまたがる再現可能な手順を説明します。
- GitHub Actionsのデプロイ制御は、環境、ジョブ、保護規則の仕様です。

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

「継続的デリバリーとは？」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/continuous-delivery/