Engineer

ブルーグリーンデプロイ

◐ 同じ構成の二つの環境の間で本番トラフィックを切り替える方法

どのようなものか

ブルーグリーンデプロイは、できる限り同じ構成にした二つの本番環境で、ソフトウェアをリリースする方法です。ここでblueと呼ぶ一方が、すべての要求を処理します。新しいバージョンは、もう一方のgreenにインストールし、顧客のトラフィックが届かない間にテストします。準備ができたら、二つの環境の前段にある仕組みを変更し、以後のすべての要求をgreenへ送ります。blueは手を加えず稼働を続けるため、greenに問題があれば、同じ変更を逆に行うだけでblueを再び使えます。

色自体に意味がないことが、この名前の狙いです。「旧・新」「主系・待機系」といった名前は、リリースのたびに意味が入れ替わりますが、二つの色なら、単に交代するだけです。次に切り替えた後はgreenが稼働側、blueが待機側になります。Martin Fowlerは、2010年に自身のblikiでこの手法を説明し、やや曖昧ながら、命名をDaniel Terhorst-NorthとJez Humbleによるものとしています。英語では通常ハイフンを入れて書きますが、ツールでは表記が異なります。この種のデプロイ戦略を追加するKubernetesコントローラー、Argo Rolloutsは、設定内で `blueGreen` と呼びます。

切り替えには、要求の送り先を決める一点であれば、どのような仕組みでも使えます。Fowlerの説明ではルーターを変更しますが、ロードバランサーの転送先、リバースプロキシの転送先、Kubernetesでトラフィックを受けるPodをラベルセレクターで選ぶServiceでも構いません。また、二つの環境を、二つのデータセンターに置く必要もありません。Fowlerは、別々のハードウェア、別々の仮想マシン、または一つのシステム上でそれぞれのIPアドレスを持つ二つのゾーンでも十分としています。

近い二つの手法と比べると、特徴が明確になります。ローリングアップデートは、インスタンスを数個ずつ置き換えるため、最後の旧版がなくなるまで、新旧が並んで要求に答えます。元へ戻すにも、逆向きに同じ更新を進める必要があり

ます。カナリアリリースは、まず少量のトラフィックを新版へ送り、段階的に広げます。ブルーグリーンは、一度にすべてを移す一方、旧版の完全なコピーを稼働したまま残すため、戻す操作も、進める操作と同じく、一段階の切り替えで済みます。

概念を判断に生かす

「ブルーグリーンデプロイとは？」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

リリースは、時間のかかる一方通行の作業から、短時間で完了し、元に戻せる作業に変わります。ファイルのコピー、プロセスの起動、キャッシュの準備といった遅い部分は、blueが処理を続ける間にgreenで行うため、顧客を待たせません。切り替え自体も、それを元へ戻すことも短時間で済みます。旧版は停止していないため、再構築も再インストールも不要です。その結果、普段の午後にリリースする際のチームの構えも、問題の兆候が見えた際に動ける速さも変わります。

顧客へ見せる前に、本番に非常に近い条件でgreenを確認することもできます。ネットワーク、設定、データベースが同じだからです。greenへのスモークテストは、細かな違いがいくつもあるステージング環境での同じテストより、多くを確認できます。Argo Rolloutsは、昇格前の新版へトラフィックを送るプレビュー用Serviceとして、これを仕組みにしています。本番のトラフィックを処理させずにテストできます。

待機側の環境には、もう一つ仕事があります。リリースが安定したら、トラフィックを受けなくなった側が、次のリリースを準備する場所になり、その間はウォームスタンバイになります。Fowlerは、切り替えのたびに人が見守るフェイルオーバーを行うため、同じ仕組みで「リリースのたびに災害復旧手順をテストできる」と指摘しています。

代償は明確です。二つの本番環境には、少なくともリリースの間、二環境分の処理能力が必要です。また、アプリケーション層は容易に複製できても、データはそうではありません。blueとgreenは、ほぼ常に一つのデータベースを共有する

ため、どのリリースでも両方のバージョンからデータを読める必要があります。それを保証できないチームは、素早く切り替えて進めても、戻す際に動かなくなります。

実際の活用例

Kubernetesでの最小構成は、二つのDeploymentと一つのServiceです。Deploymentの `web-blue` と `web-green` が同じアプリケーションを動かし、 `app: web` と、 `slot: blue` または `slot: green` のラベルを持ちます。顧客がアクセスするServiceは、一方のスロットを選びます。

```
apiVersion: v1
kind: Service
metadata:
  name: web
spec:
  selector:
    app: web
    slot: blue
  ports:
    - port: 80
      targetPort: 8080
```

リリースでは、待機側に新しいイメージを配置し、準備が整うのを待ち、 `slot: green` だけを選ぶ二つ目のServiceでテストしてから、稼働側のServiceを切り替えます。

```
kubectl set image deployment/web-green web=registry.example.com/web:1.5.0
kubectl rollout status deployment/web-green --timeout=5m
# ここでweb-greenのプレビュー用Serviceにスモークテストを行う
kubectl patch service web -p '{"spec":{"selector":{"slot":"green"}}}'
# 切り戻しは、色だけを変えた同じコマンドで行う
kubectl patch service web -p '{"spec":{"selector":{"slot":"blue"}}}'
```

`kubectl patch` の既定はstrategic merge patchであるため、セレクターの `slot` キーだけが変わり、 `app: web` はそのまま残ります。Serviceコントローラーは、セレクターに一致するPodを継続的に調べ、その集合が変わるとServiceのエンドポイントを更新します。そのため、新しい接続はgreenへ届き、blueは旧イメージで待機を続けます。

Argo Rolloutsは、同じ一連の処理を、一つの `Rollout` オブジェクトで自動化します。昇格時に新しいPodへ向ける `activeService` と、それより前にアクセスするための、任意の `previewService` を使います。 `autoPromotionEnabled: false` にすると、誰かが `kubectl argo rollouts promote web` を実行するまで、切り替え前でロールアウトを一時停止します。

DevOps・CI/CDサービスは、変更をテストし、リリースし、元へ戻せることを約束しています。ブルーグリーンデプロイは、最後の操作の負担を小さくする方法の一つです。その基盤になるのは、あるプラットフォームのAzureへのデプロイで、二つ目の本番スロットを設けずに実践している、二つの習慣です。Bicepテンプレートで環境間の対象インフラを同じ形に保ち、同じビルド成果物を開発からステージング、本番へ昇格させ、到着した環境の設定を取得します。その経路は、リリースを各環境へ直線的に進めます。ブルーグリーンは、その終点に双子を加えます。インフラを同じにし、一つの成果物を変えずに使うという二つの習慣が、その双子が本当に同じだと信頼できる根拠になります。

問題が起きるところ

スキーマの変更で、戻る道が行き止まりになります。 greenが起動し、列名を変えるマイグレーションを実行して、正常にトラフィックを処理します。10分後、障害でblueへ戻すと、旧列名を使い続けるため、すべての要求が失敗します。アプリケーションのロールバックは計画しても、データの計画がなかったのです。Fowlerの2015年の助言は、二つを分けることです。両方のバージョンで使えるデータベースの変更を先に出し、その後アプリケーションを出し、新版が安定してから、旧版だけに必要だったものを削除します。

greenが書いた状態が、切り戻すと取り残されます。 ローカルディスクにアップロードしたファイル、メモリー内のセッションストア、保留中ジョブの独自キャッシュなど、greenだけが持つものは、blueへトラフィックを戻しても、そちらにはありません。利用者は、ログアウトさせられたり、保存したばかりのものを失ったりします。共有する状態は、両方の色が使う保存先に置きます。greenに固有のデータが必要なら、Fowlerは、両環境へトランザクションを送るか、greenを読み取り専用で開始し、動作を確認できてから読み書き可能へ切り替える方法を提案しています。

障害が現れる前に、戻る先を縮小してしまいます。 Argo Rolloutsは、切り替え後に旧ReplicaSetを縮小し、 `scaleDownDelaySeconds` の既定値は30です。15分後や夜間ジョブの実行時、あるいはゆっくり進むリークによって障害が表れたときには、戻るblueがありません。通常、問題が表れるまでの時間をカバーするよう遅延を設定し、その代償として処理能力を確保し続けることを受け入れます。

```
strategy:
  blueGreen:
    activeService: web
    previewService: web-preview
    autoPromotionEnabled: false
    scaleDownDelaySeconds: 3600
```

準備完了だけで、正しさを確かめずに昇格します。 `autoPromotionEnabled` の既定値はtrueであり、新しいPodが準備完了になると、Argo Rolloutsは切り替えます。readiness probe（準備完了の確認）で分かるのは、プロセスが応答することで、購入手続きが動くことではありません。上のブロックのように昇格前で一時停止し、プレビュー用Serviceでテストします。または、失敗時にロールアウトを中止する `prePromotionAnalysis` と、切り替え後に失敗すると旧版へ戻す `postPromotionAnalysis` を付けます。

待機側の構成がずれます。 リリースの間に、誰かが緊急の設定変更を稼働側だけへ適用します。次のリリースを置く待機側は、古い設定のままで、一つだけ変えるはずだった切り替えが、二つを変えます。両方のスロットを同じテンプレートと設定の正本から構築し、一方の変更が両方に反映されるようにします。Fowlerの説明のとおり、待機側を次のリリースの準備場所として扱います。

参考資料

- Blue Green Deployment：Martin Fowlerによる2010年の説明と、データベース変更に関する2015年の追記です。
- Argo Rollouts：BlueGreen：稼働用とプレビュー用のService、自動昇格、切り替え前後の分析、縮小までの遅延を説明します。
- KubernetesのService：Serviceのトラフィックを受けるPodを、セレクターで選ぶ仕組みです。
- kubectl patch：セレクターを切り替えるコマンドと、strategic merge、merge、JSONの各パッチを説明します。

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

「ブルーグリーンデプロイとは？」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/blue-green-deployment/