CI/CDを使い倒して数段上のソフトウェア開発しよう

#devsumi #circlecijp

CI/CD 戦国時代

Google GCP Cloud Build

Google Cloud　Google を選ぶ理由　サービス　ソリューション　料金　セキュリティ　ドキュメント

デベロッパー ツール

Cloud Build

継続的なビルド、テスト、デプロイを実現します。

ドキュメントを見る　コンソールへ移動

スピードと柔軟性

Cloud Build では、すべての言語でソフトウェアを迅速にビルドでき、複数の環境（VM、サーバーレス、Kubernetes、Firebase など）でのビルド、テスト、デプロイ用にカスタム ワークフローの定義を完全に制御できます。

AWS CodeBuild

aws　日本担当チームへお問い合わせ　サポート　日本語

製品　ソリューション　料金　ドキュメント　学習　パートナー　AWS Marketplace　その他

AWS CodeBuild　概要　特徴　料金　開始方法　よくある質問　製品の統合　お客様

AWS CodeBuild

継続的スケーリングによるコードのビルドとテスト
費用は構築する回数分のみ

AWS CodeBuild の使用を開始する

AWS CodeBuild は、ソースコードをコンパイルし、テストを実行し、デプロイ可能なソフトウェアパッケージを作成できる完全マネージド型のビルドサービスです。CodeBuild により、ビルドサーバーのプロビジョニング、管理、スケーリングが不要になります。CodeBuild は連続的にスケールされ、複数のビルドが同時に処理されるので、ビルドが待機状態でキュー内に残されることがありません。パッケージ済みのビルド環境で、すぐに開始できます。自分のビルドツールを使用するために、カスタムビルド環境を作成することもできます。CodeBuild では、コンピューティングリソースの使用に対して、分単位で料金が発生します。

Microsoft Azure Pipelines

Microsoft Azure　営業担当者へのお問い合わせ: 0120-337-499　検索　アカウント　ポータル　サインイン

概要　ソリューション　製品　ドキュメント　価格　トレーニング　Marketplace　パートナー　サポート　ブログ　その他

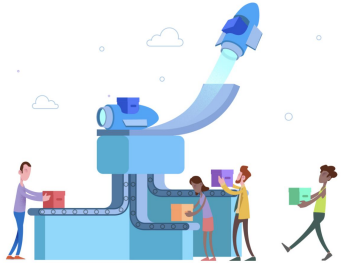
Azure DevOps Services / Azure Pipelines

Azure Pipelines

どのようなプラットフォームやクラウドにも継続的にビルド、テスト、デプロイできます

Pipelines の使用を無料で開始する

アカウントを既にお持ちですか?
Azure DevOps にサインイン >



CI/CD 戦国時代


Europe wercker
WerckerをOracleが買収、コンテナベースのデベロッパープラットホームに既存大手も着目
2017年4月18日 by Mike Butcher


developers
CloudBees
TC
developer
JenkinsによるCI/CDの自動化サービスを提供するCloudBeesが$62Mを調達、ますます買収志向に


ネットサービス
M&A
exit
developer
Cloud
SQLツールの古参IderaがTravis CIを買収して継続的インテグレーションをメニューに加える
2019年1月24日 by Hiroshi Iwatani

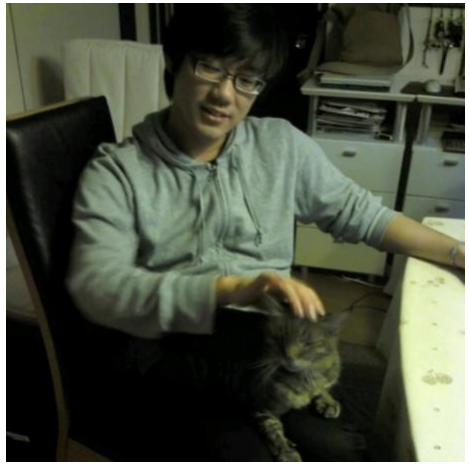
最初の疑問

なぜCI/CDへの関心がこれほど高まっているのか？

自己紹介:

Kim, Hirokuni (金 洋国)

- 元CircleCI 開発者
- CircleCI Japan Tech Lead



”この発言は個人の見解ではなく所属する組織を代表しています!!”

モチベーションについて

このセッションの基本の流れ

What

Why

Why Not

Beyond

宣伝 (会社)

CircleCI初の海外支社

- **日本語サポート**
- ドキュメントの日本語化
- ユーザーコミュニティー

@CircleCIJapan

FB Community Group

宣伝 (個人)

電動キックボードを体験できるサービス Hop-on! を運営

- 日本で唯一のサービス(のはず)
- みなとみらいで体験できます
- 続きはWebで！

CI / 継続的インテグレーション

What is CI?

その前に確認

もちろんテストは書いてますよね？

なぜテストを書くべきか

- 何度も同じ手順を繰り返さないといけない
- 人の目や手に頼ると必ず見落としが発生する

なぜテストを書くべきか

- 何度も同じ手順を繰り返さないといけない
- 人の目や手に頼ると必ず見落としが発生する

それコンピューターにやらせようよ！

CIとはテストを自動で実行する仕組み

開発者のコード変更に対して

- 常に
- 同じ環境で

テストを実行してくれる

Why CI?

ただテストを書くだけでは不十分

- テストがあるけど実行し忘れた
- 昔書いたテストが壊れていて動かない
- テスト結果が環境依存

ただテストを書くだけでは不十分

- テストがあるけど実行し忘れた
- 昔書いたテストが壊れていて動かない
- テスト結果が環境依存

テストの信頼性がない

問題: テストの実行忘れ

- リリース前にテストをしわすれる
- バグを見落とす
- 修正でリリースが遅れる

解答: 常にテストを回す

- GitHub (VCS)の変更をCI/CDが検知
- 全変更に対してテスト実行

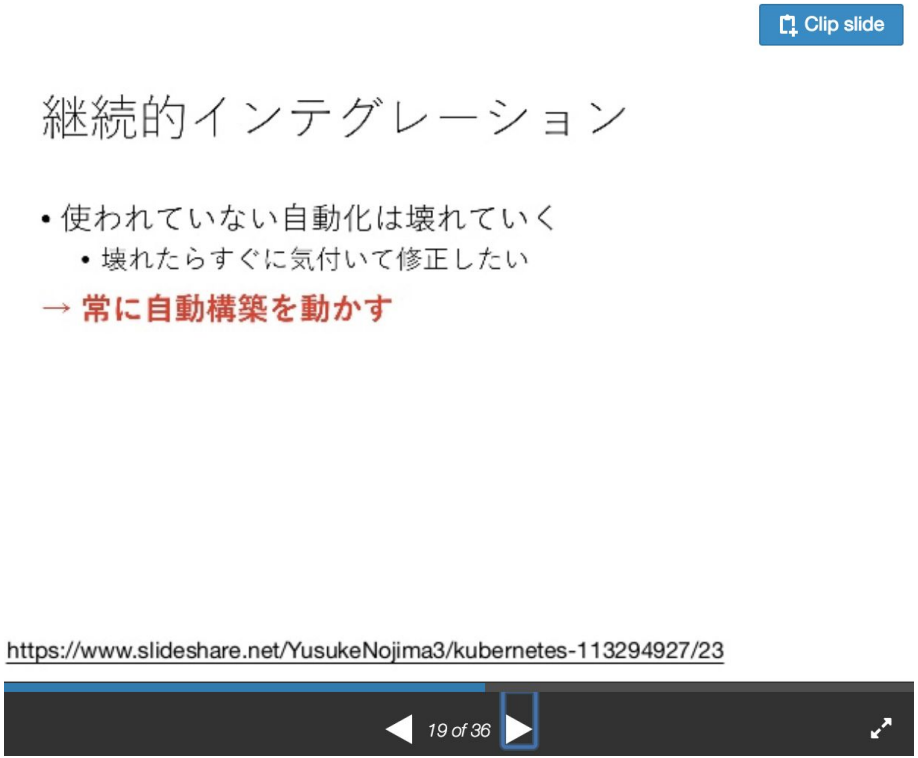
問題: テストが壊れてしまう

- 古いテストが壊れている
- テストが悪いのかコードがわからない

解答: 壊れたテストを素早く検知

- テストが壊れた時点で検知
- 直さないとマージできない (後述）

使われていない自動化は壊れていく



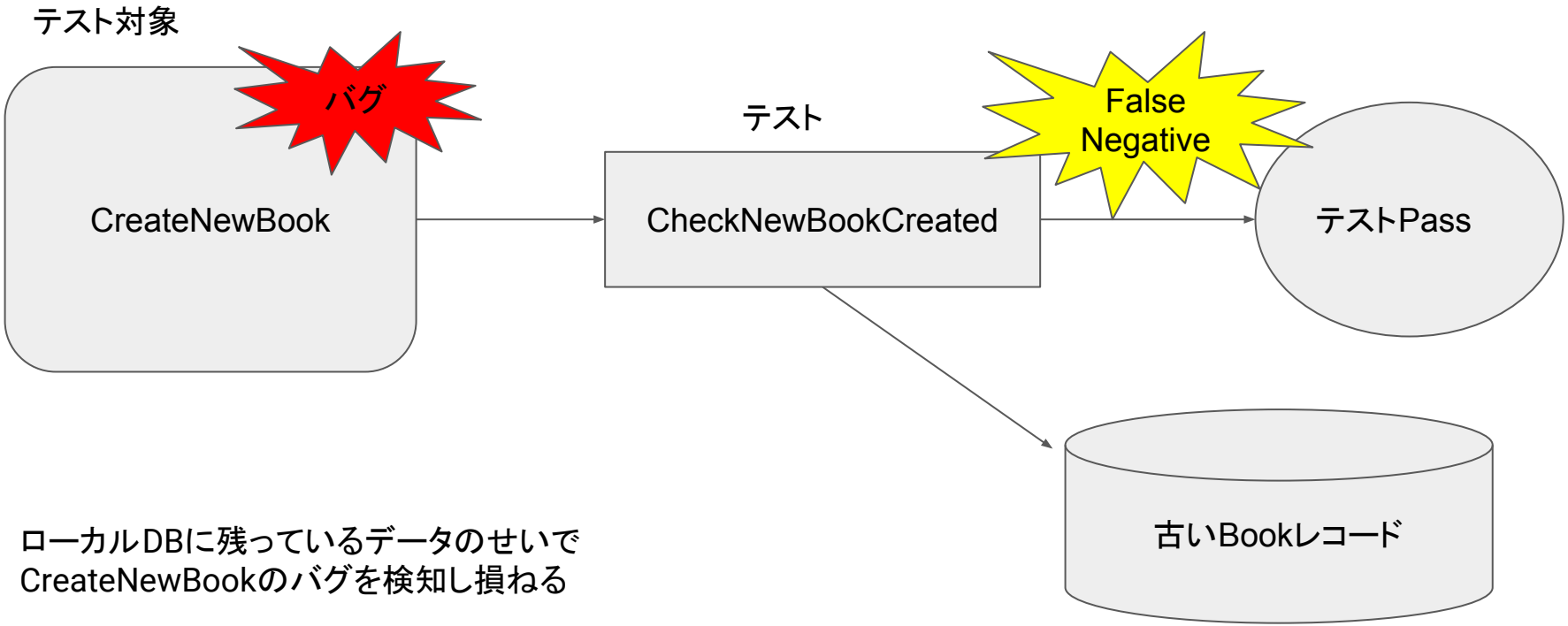
“サイボウズを支えるCircleCI”より

問題: テスト結果が環境依存

僕のマシンだとテスト通ってます

(`・ω・´) ｷﾘｯ

例: テスト環境の差異による問題



ローカルDBに残っているデータのせいで
CreateNewBookのバグを検知し損ねる

解答: CIを唯一のテスト環境にする

- 毎回同じテスト環境が構築される
- まっさらな環境でテストを実行
- いつ実行しても同じテスト結果になる

CIの目的

テストの新陳代謝を高めて信頼性をあげる

Why NOT CI?

CI導入を妨げる問題

- テストがない
- メンテナンス

問題: テストがない

CIを導入する上でいちばんやっかいな問題

- CIを始めたい
- でも実行するテストが存在しない
- テスト文化の布教にはコストと時間がかかる

テストがない状態からCIを始める5ステップ

Step 1: お好みのCI/CDツールを選ぶ

Step 2: タスクの自動化

Step 3: 可視化する

Step 4: マージブロック

Step 5: テストを追加していく

Step 1: お好みのCI/CDツールを選ぶ

ステップ2: 様々なタスクを自動化しよう

テスト以外のタスクを自動化

- 構文チェック（linting)
- カバレッジ計測
- 循環的複雑度のチェック
- ドキュメントの自動生成

ステップ3: 可視化しよう

CI結果を可視化しよう

- ステータスバッジ
- ダッシュボードの作成
- メール・チャットでの通知

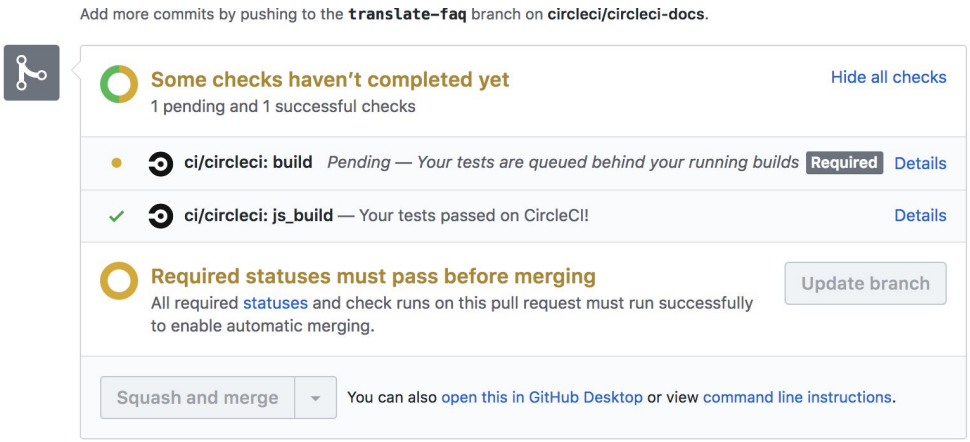
CIやってる感が出てくる

”お、俺たちCIしてるっぽいw”

Step 4: マージブロック有効化

マージするための条件をブランチごとに
指定できる機能

- CIが通らないとマージできない
- 管理者しかマージできない
- Force Push禁止
- Etc, etc



CircleCIが通らないとマージできない

Step 5: テストの追加

少しずつテストを追加していく

- ユニットテストはとりあえず後回し
- 最も大事なビジネスロジック
- この時点では無理は禁物....

CI導入を妨げる問題

- テストがない
- メンテナンス

問題: メンテナンス

- CI/CDツールのメンテナンスは大変
- 通常専任のエンジニアが必要
- CircleCIのようなクラウド型がおすすめ

クラウド型 VS オンプレミス型

クラウド型

AWS CodeBuild

CircleCI

GCP Cloud Build

Travis CI

オンプレミス型

Jenkins

Concourse CI

CIのまとめ

- テストの信頼性と品質を向上させる
- テストがなくてもCIは始めれる
- できる自動化からはじめよう
- クラウド型のツールで運用コストを下げる

Beyond CI

開発フロー

コードをPush

開発フロー

コードをPush

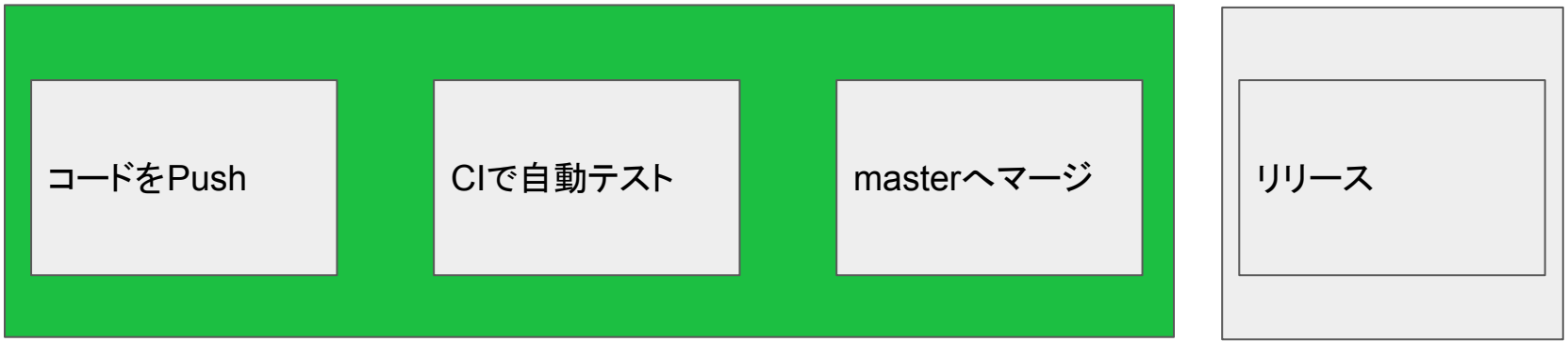
CIでテスト

開発フロー

開発フロー

自動

開発フロー



CD / 継続的デプロイメント

What is CD?

CDとは？

Continuous Delivery (継続的デリバリー)

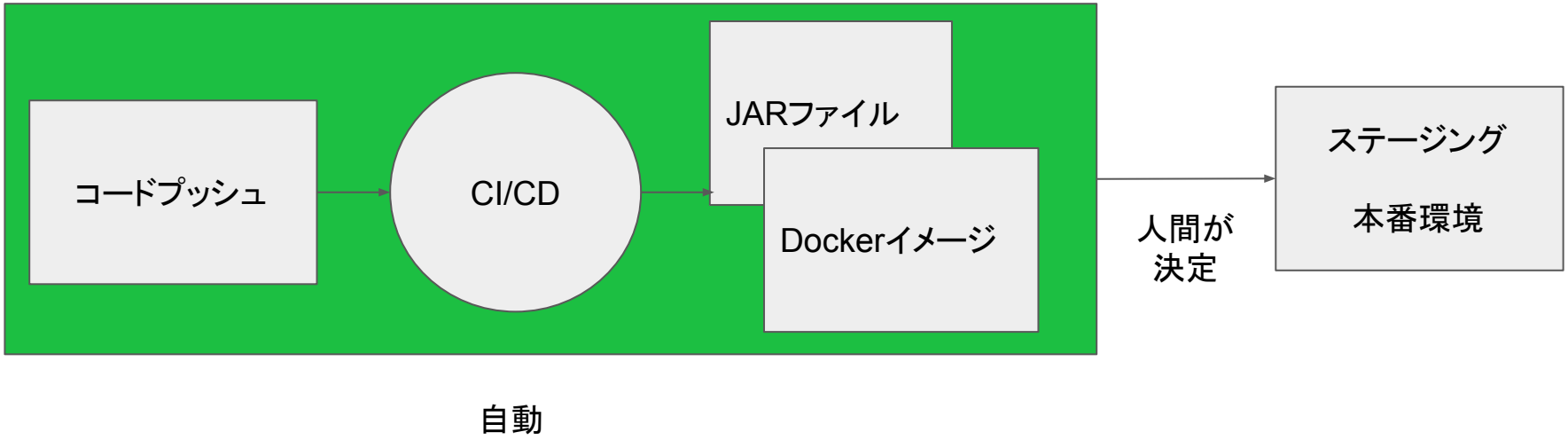
常にリリース可能な状態を維持する

Continuous Deployment (継続的デプロイメント)

自動でステージング・本番環境へデプロイ

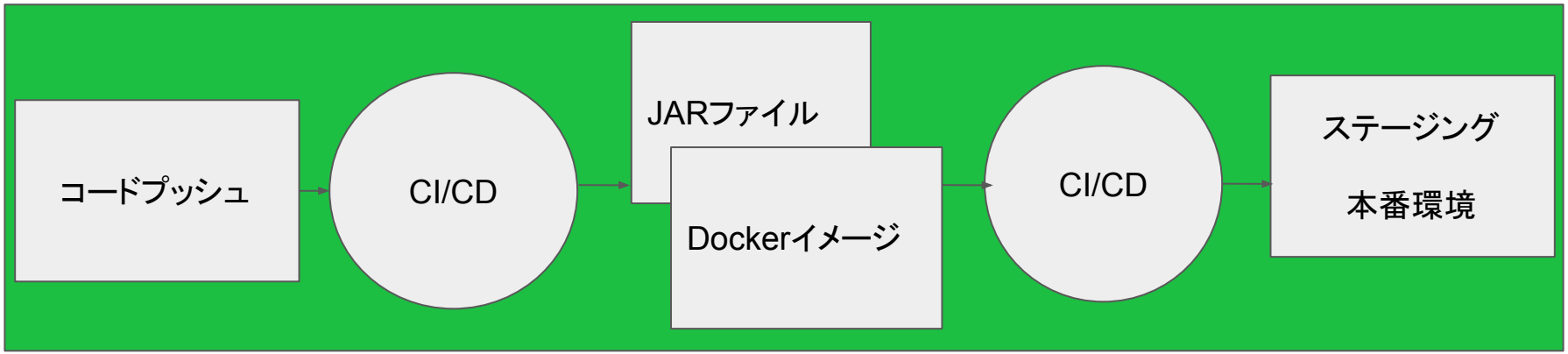
Continuous Delivery (継続的デリバリー)

リリース作業に人間の意思が介在する



Continuous Deployment (継続的デプロイメント)

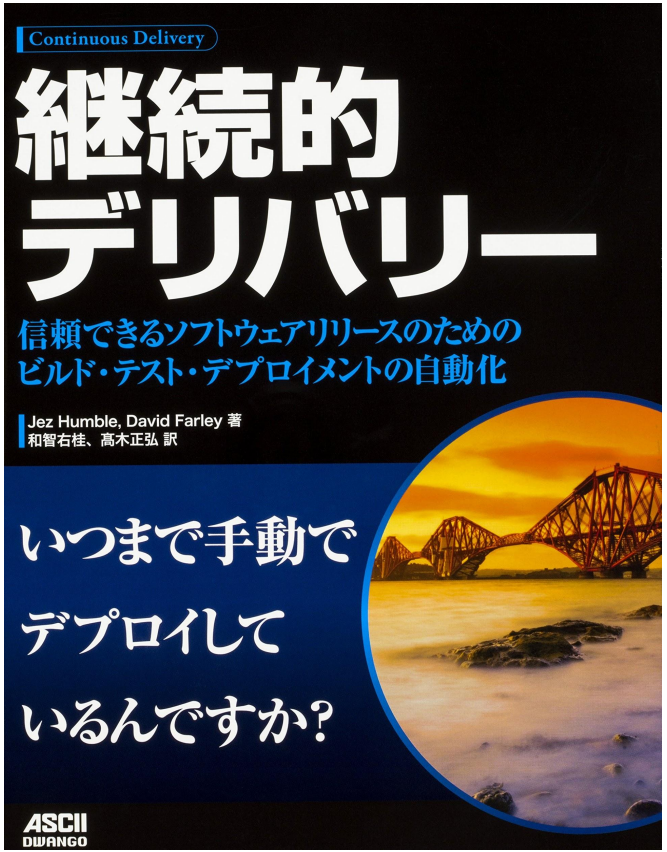
リリースに人の意思が介在しない



全自動

広義の継続的デリバリー

ビジネス価値を継続的に
デリバリーしていくこと



デプロイとリリースの違い

デプロイ

コードを本番環境に配置すること

リリース

配置したコードでトラフィックをさばくこと

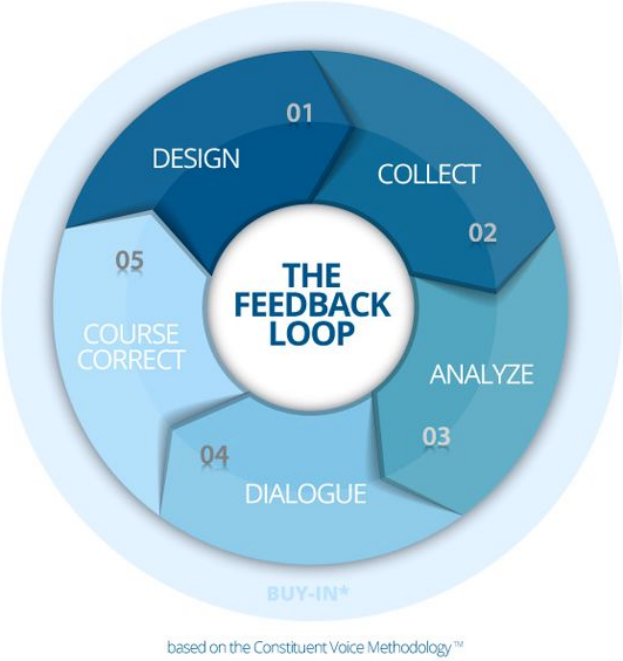
Why CD?

よくあるリリース後の問題

- 検証環境で見つからなかったバグ
- 仕様と全然違う動きをする
- そもそも仕様が間違ってた

解答: フィードバックループを使おう

- 細かい単位でリリースする
- フィードバックを早めに得る
- カイゼンする



CIなしだとだとループが回らない

- リリースの許可が必要
- ヒューマンエラー

フィードバックループが回らない

CDなくしてフィードバックループなし

No CD, No Feedback Loop

最初の疑問

なぜCI/CDへの関心がこれほど高まっているのか？

Why NOT CD?

Why NOT CD?

CDを始める上での2つの問題

- 技術的な問題
- 組織的な問題

CD導入の技術的な問題

そもそもアーキテクチャーがCDに向いてない

- エンタープライズなアーキテクチャー
- レガシーなアーキテクチャー

解答: CD導入の技術的な問題

時間をかけてアップデート

- サービスの疎結合
- 徐々にモダン化

CD導入の組織的な問題

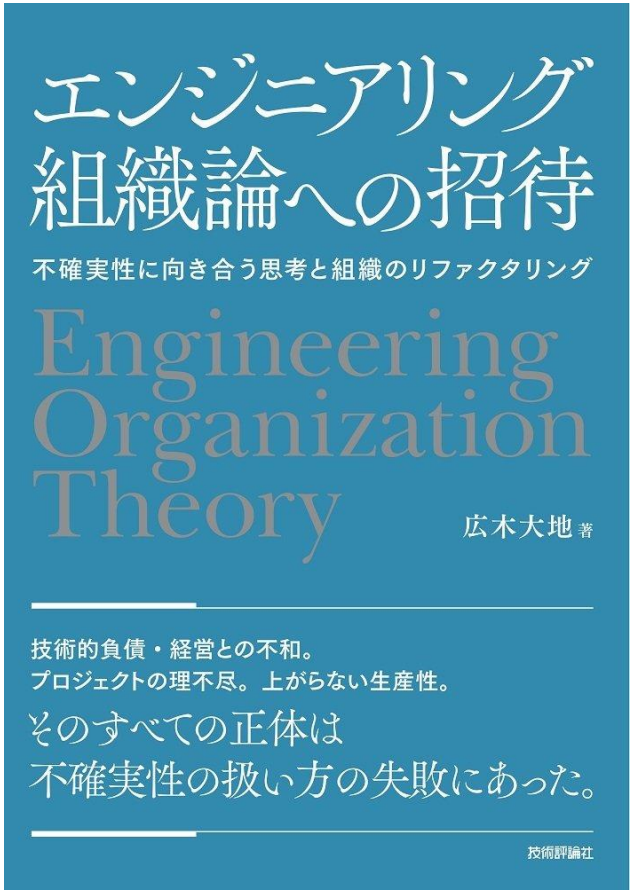
~~CDするには不向きな組織~~

- ~~官僚的な組織~~
- ~~失敗に対する許容が低い組織~~

解答: CD導入の組織的な問題

誰か教えてください....

組織の問題に関してはこれ呼んでください！



新システムにはまずCI/CDを導入しよう

家永 英治さんのブログより

CircleCIでの事例

Before:

- 常に200台以上のビルドマシンからなるフリート
- Chat Ops (hubot)でデプロイ
- およそ2日で完全に入れ替わる
- しばらく古いコードと新しいコードが混在する問題

CircleCIでの事例

1年かけて以下を実施した

- DockerとKubernetesの導入
- マイクロサービス化

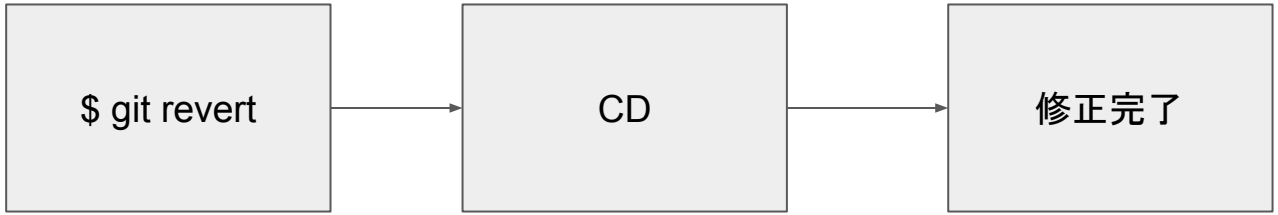
CDのまとめ

- CDが回るとフィードバックループも回る
- CDに向いていない技術・組織はある
- 既存システムに導入が無理なら新システムから

Beyond CD

CI/CD完全に理解した、でしょうか？

CDのその先1: 迅速なロールバック



CDのその先2: 本番環境でのテスト

テスト環境での失敗例

１週間テスト環境でテスト

テスト失敗例

１週間テスト環境でテスト

↓

リリース (完璧だ!)

テスト失敗例

１週間テスト環境でテスト

↓

リリース (完璧だ!)

↓

本番環境のDockerのバージョンが古くて
バグを踏む 😣

テスト失敗例

Dockerのバージョンも同じにした！

テスト失敗例

Dockerのバージョンも同じにした！

↓

リリース (今度こそ完璧だ!)

例 2

Dockerのバージョンも同じにした！

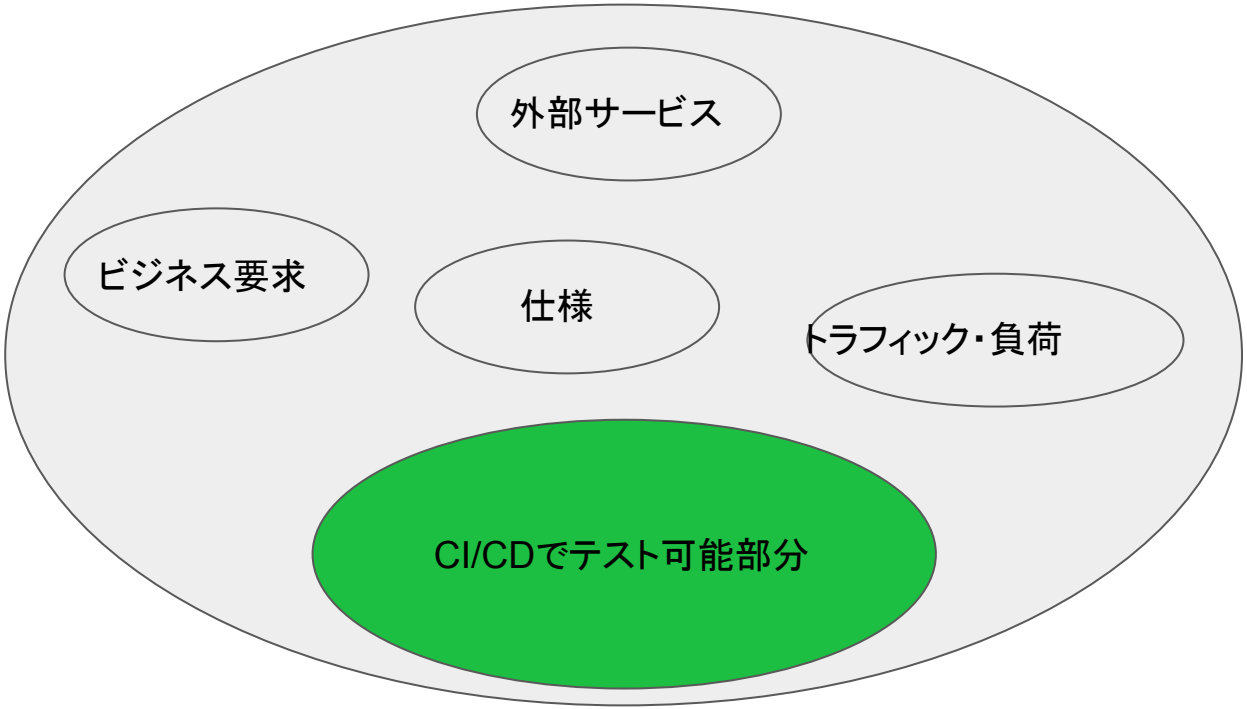
↓

リリース (今度こそ完璧だ!)

↓

GitHubのAPI RateLimitにひっかかる 😭

なぜこんなことが起こるか？



テスト可能な部分はとても小さい！

僕たちの重大な学び

リリースしてみないと結局わからない！

CDのその先3: 高度なリリース手法

- カナリーリリース
- ブルー グリーン デプロイ

CDがもたらす心理的安全性

- 迅速なロールバック
- 本番環境でのテスト
- 高度なリリース手法

これらがもたらすものは、、、

CDがもたらす心理的安全性

- 迅速なロールバック
- 本番環境でのテスト
- 高度なリリース手法

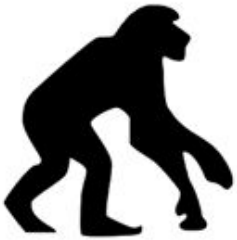
これらがもたらすものは、、、

Why CD?

CI/CD Makes Programming FUN!!

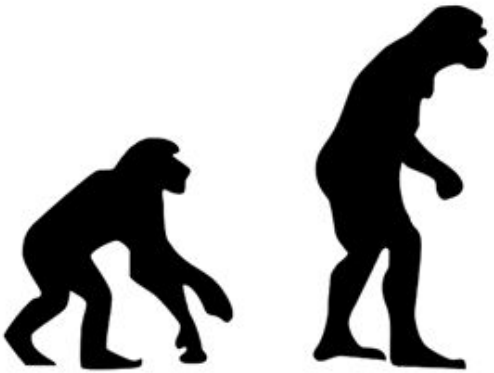
CI/CDの未来

CI/CDはどこへ向かうのか？



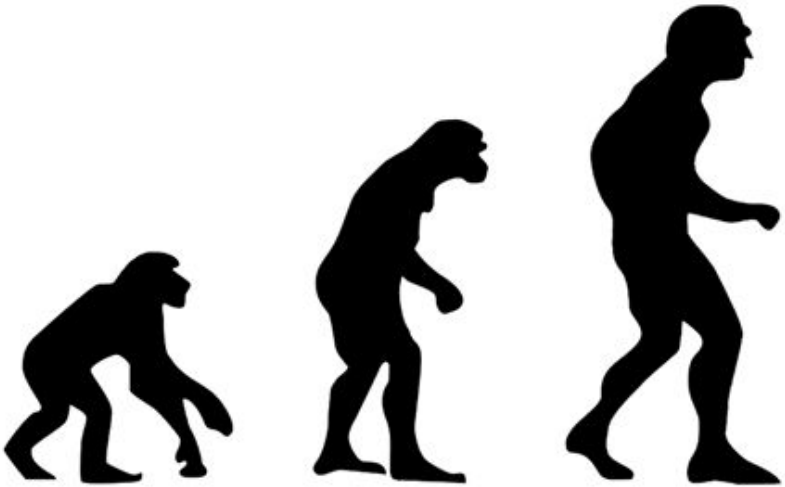
CI

CI/CDはどこへ向かうのか？



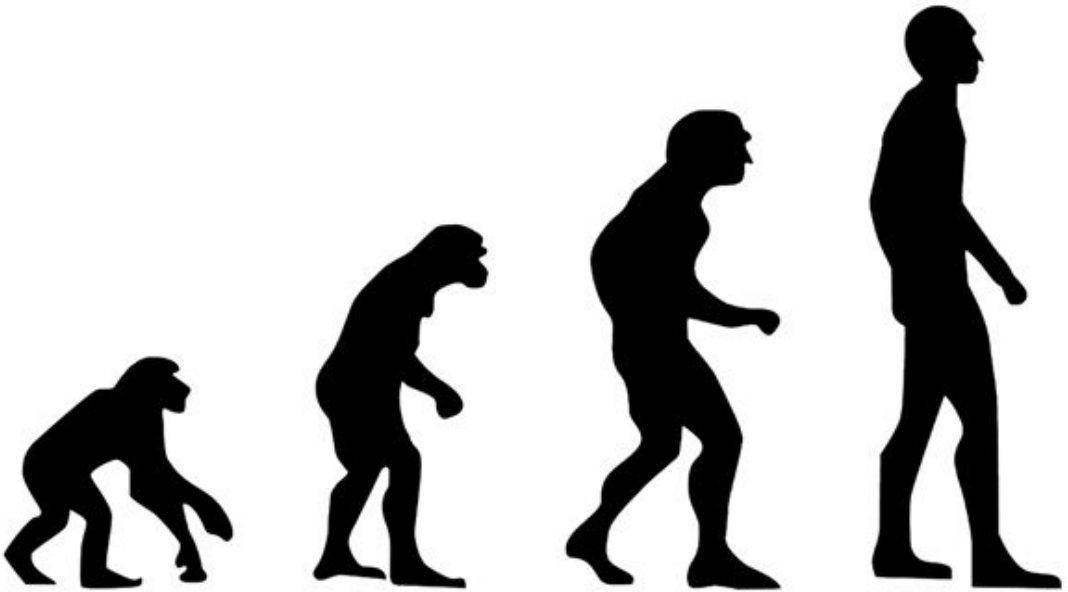
CI　　CDelivery

CI/CDはどこへ向かうのか？



CI　　CDelivery　　CDeployment

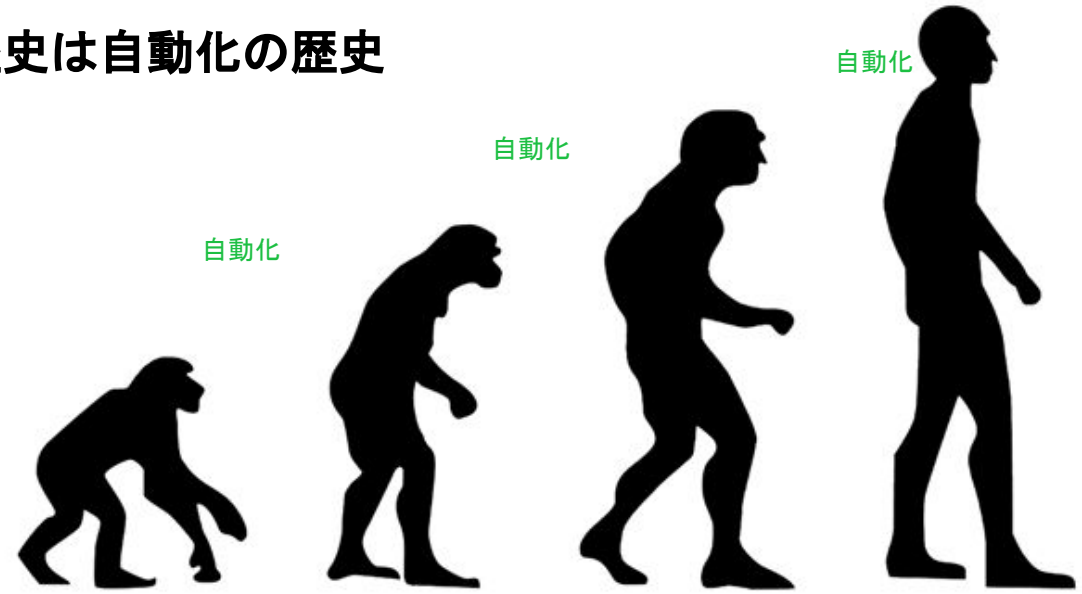
CI/CDはどこへ向かうのか？



CI　　CDelivery　　CDeployment　　????

CI/CDはどこへ向かうのか？

CI/CDの歴史は自動化の歴史



確実な自動化の未来

今手動でやっていることを意識しなくてもいい時代

- CIやCDの設定
- モニタリング
- デプロイ環境の構築

CI/CDの未来

$ git commit -m “First commit” && git push

最初からクライマックス!!

CircleCI ユーザーコミュニティーのご紹介

FB Community Group

Thank you.