CI/CDとGitLabでのイシュー（issue）

J23072　蝦名武尊

# はじめに

このドキュメントの目的はGitHubやGitLabのCI（継続的インテグレーション）/CD（継続的デリバリー）についてまとめることである。また、GitLabでのイシュー(issue)について書き方をまとめることも目的の一つである。

# CI/CD

## CI/CDとは

　CI/CDとは継続的インテグレーション（Continuous Integration）と継続的デリバリー（Continuous Delivery）の略称である。

　CIとは開発者がコードを修正しプッシュした際に自動的にビルドやテストを実行するプロセスや仕組みのことである。これにより、継続的なテストを行い、コードの品質を保つことができる。

　CDは、CIによる統合や一体化を拡張したものであり、より高度にかつ自動的にシステムの変更が反映されるように設計された環境や概念のことである。

　CI/CDの目的は分割されているツールやプラットフォームを統合し、変更が自動的に他ツールに反映されるように環境を組むことで開発の効率化を図ることだ。ツールやプラットフォームを指すこともあれば、手法や概念として用いられることもある。

## メリット

　CI/CDのメリットは大きく分けて4つである。

1. 制作物の品質が高まる

CI/CDを利用することで制作物の品質が高まる。従来であれば、時間のかかる最新のコードやテクノロジーの適用を容易に行えるようになり、開発環境を最新に保つことができるため、最先端・高品質の制作が可能となる。

1. 開発スピードが向上する

　CI/CDはツール間で連携されており、入力情報が他ツールに自動的に適用されるため、作業量が減少できる。これにより、開発スピードの向上とともに、急な変化への対応が容易になる。

1. 開発環境の差異による問題を予防できる

　CI/CDでは各ツールが自動的に提携されており、環境の際による問題が予防でき、環境ごとの仕様が原因のエラーやセキュリティ問題が減少する。

1. 開発者が開発に専念できる

　CI/CDではエラーが発生しにくいため、開発者がエラー対応に充てていた時間を開発に注ぐことができる。

## ベストプラクティス

　新しくCI/CDを導入する場合、以下のベストプラクティスを実施することで効果をより高めることができる。

### コードの品質管理

CI/CDは、ビルドやテスト、デプロイまで同じコードを用いて開発を進める。そのため、初期コードで間違いがある場合には影響が後半にまで及ぶ。CI/CDでは通常の開発以上にコードの適切な管理が必要である。

開発者間での認識の齟齬が生じないように、コートレポジトリの管理や定期的なコードレビュー、コード変更の際のレポジトリの書き換えを必ず行うこと。

### 初期は微調整を頻繁に

　CI/CDは一度組織に馴染むと開発スピード向上やエラー削減に大きく貢献する。新しく導入したCI/CDを最適化するには開発者が何に困っているかや、どの部分がうまく動作しないなどの声を拾い上げ、必要に応じてCI/CDを微調整する必要がある。

### 適切なセキュリティ対策を

　CI/CDはビルド、テスト、デプロイが１つの環境で行える反面、情報漏洩の影響が制作物すべてに及ぶ可能性がある。そのため、ログイン情報の秘匿や認証の強化、問題発生時のリスクマネジメントを制定しておくなど、セキュリティ対策はより強固に行う必要がある。

### 2.3.4 設定ファイルを再利用する

　CI/CDは設定ファイルに記述されたコードを再利用することで、開発の効率化を可能にしている。しかし、開発者の人数が増えると情報共有が困難になり設定ファイルの情報を生かせずに開発が進行してしまうことがあるため、組織として設定ファイルの活用法や設置場所を共有し対策を図るべきである。

### 2.4.5エラーを放置しない

　CI/CDはコードを複数個所に反映させることで高速化とエラー減少を実現している。そのため、初期に利用したコードにエラーが存在していた場合、影響がその他の箇所にも及ぼす。故にエラーに気づき次第、修正を図るべきである。

### 2.3.6 デプロイごとに本番前環境をクリーンアップする

　環境の実行期間が長くなるほど、適用された構成変更や更新のッ追跡が困難になる。CI/CDではデプロイごとに本番前環境をクリーンアップすることで以前のデータの悪影響を防ぐことができる。

## 評価方法

　導入したCI/CDの成果を評価する場合には、以下の項目を用いて成果を検討するべきである。

### サイクルタイム

　サイクルタイムとは１つの作業工程の開始から完了までにかかる時間であり、今回はコードの書き始めからアプリ完成までの時間である。これをCI/CDの導入前後で比較する。

### 作業時間

　開発者が何にどれだけの時間を費やしているかを記録、分析する。CI/CDを導入後、問題の処理やツール間の調整にかかる時間が減り、開発に集中できているようであればCI/CDがうまく機能しているという評価を下すことができる。

### エラー率

　CI/CD導入前後でエラー率に変化があるかを調査する。もし、CI/CD導入後にエラーが多発あるいは修正にかける時間が削減されていない場合は、CI/CDがうまく機能していない可能性が存在する。

* + 1. インフラコスト

　CI/CD導入にかかる費用はもちろん、コストパフォーマンスが適切であるかどうか検討する。CI/CD導入にはメリットがある反面、インフラストラクチャ構築費用が掛かるため、開発スピードやエラー率、人件費などと比較しコスト面から導入が妥当か評価する。

### 開発者のレビュー

　実際に開発を行う現場のレビューも重要な評価基準である。ツールの使いやすさや、体感的なエラーの数、開発者同士の連携などの質問により、CI/CDの開発者目線の評価を得ることができる。

## CI/CDパイプライン

　CI/CDは様々なステージや環境、ツールなどをパイプラインで結ぶことで自動化を図り作業の効率化を実現する。例えば、通常独立している「ビルド」「テスト」「デプロイ」の各ステージを結ぶことで特定の変更が他のステージにも反映される。CI/CDパイプラインはCI/CDが効果を発揮する上で核となる構造である。

　パイプラインが正常に機能しているプラットフォームやツールを使用することでCI/CDを最大限活用することができる。しかし、CI/CDパイプラインが正常に機能しない場合、CI/CDの恩恵を受けられないばかりか、エラーの増加や作業時間の増加が起こるため、CI/CDツールの選定は慎重に行う必要がある。

# GitLabでのイシュー

GitLab上においてイシュー（issue）とはタスクを意味する。イシューを作成する目的としてタスクの明瞭化や実装漏れの削減などが挙げられる。

## 作成方法

1. イシューを作成するプロジェクトに移動する。
2. サイドバーの「project」にある「issue」をクリックし、イシュー一覧画面委移動する。
3. 画面上の「New issue」ボタンをクリックする。
4. タイトルやタイプ、説明を記載し「Create issue」をクリックし完了である。

## 作成時の要点

イシュー作成時に意識すべきことを項目毎に記載する。

### タイトル

　タイトルは短く簡潔に要点を記載する。また、だれが読んでも内容が想像できるような明確なタイトルにするべきである。

* + 1. タイプ

　タイプについては実装時には「issue」、試験時の不具合について記載する場合は「incident」と記載する。

### 説明

　このイシューは何をする目的で作成されどのようなアクションをとるべきなのかということや問題の背景や解決で期待される効果などを記述する。

### 期限

　このイシューをいつまでに完了させなければいけないのか必ず設定する。

### ラベル

ラベルを活用してイシューのカテゴリーや優先度を示し、開発担当やバグ修正など、目的に応じたタグ付けを行う。

# 終わりに

CI/CDを正しく活用することで、エラー対応の時間削減や開発効率向上が期待できる。また、セキュリティ対策や環境整備などのベストプラクティスを取り入れることで、その効果を最大限に発揮できる。

GitLabでイシューを作成する場合はタイトルの分かりやすさやタイプの記載などを確実に行うことでタスクの明確化や記載漏れの予防を期待することができる。

# 参考文献

・GitLab合同会社.” **CI/CDとは？ベストプラクティスやメリットも解説.**”Valerie Silverthorne,2024/11/6, <https://about.gitlab.com/ja-jp/blog/2022/02/03/how-to-keep-up-with-ci-cd-best-practices/>

・セカンドセレクション.” **【初心者向け】めっちゃ簡単!?GitLab issueの作り方.**”Zenn, 2024/03/25, <https://zenn.dev/secondselection/articles/_create_issues>