

# クラシルの開発で使ってる **GitHub Actions**



クラシル Tech Talk #3

# 自己紹介

- Twitter: @penguin\_sharp
- GitHub: MeilCli
- Speaker Deck: MeilCli ← 今日のスライドはこのアカウントで公開します
- Skill: **C#**, Kotlin, Android, Azure Pipelines, **GitHub Actions**
- Career:
  - 新卒で入社した会社で Android・Xamarin.Android アプリ開発を行う
  - 2020/2 に dely に入社し クラシル、特に チラシ 機能の開発に関わる

# まえがき

クラシルではCI/CDにさまざまなサービスを使っています

AWS CodeBuild, Bitrise, GitHub Actions

# まえがき

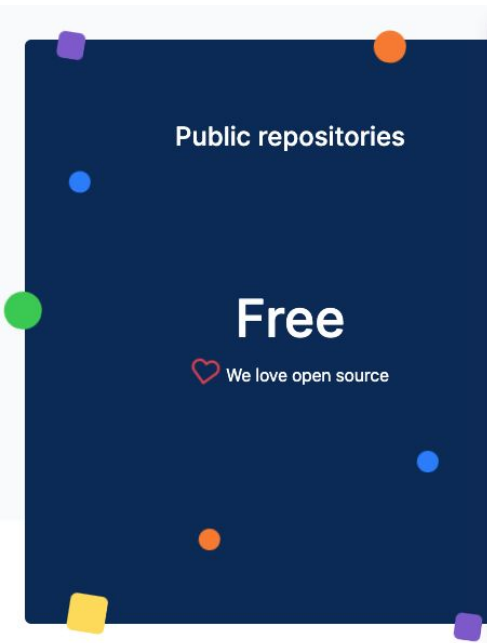
ぶっちゃけると主にAWS CodeBuild使ってます

# まえがき

今回はMeilCliが今までに~~社内のCodeBuild派に抗って~~  
導入してきたGitHub Actionsを紹介します

# GitHub Actionsのなにが いいのか

- OSSに優しい料金設計
- Windows, macOS, Linux環境
- MS資本
- ~~RunnerがC#製~~
  - <https://github.com/actions/runner>



<https://github.com/features/actions>

# GitHub Actionsのなにがいいのか

いろいろいいところあるけど

やっぱりGitHubと親和性が高いのがいい

CI/CDするだけならAzure Pipelinesで十分ですし...

# GitHub Actionsを軽くおさらい

GitHub RepositoryのTopにあるここからActionsをクリック

<> Code

! Issues

🔗 Pull requests

▶ Actions

🛡 Security

📈 Insights

⚙ Settings



# GitHub Actionsを軽くおさらい

最初はRepositoryから推測されたおすすめ Workflow がサジェストされるので問題なさそうならそれを活用する

## Get started with GitHub Actions

Build, test, and deploy your code. Make code reviews, branch management, and issue triaging work the way you want. Select a workflow template to get started.

Skip this and [set up a workflow yourself](#) →

### Workflows made for your C# repository Suggested

#### .NET Core

By GitHub Actions

Build and test a .NET Core or ASP.NET Core project.

[Set up this workflow](#)

```
dotnet restore
dotnet build --configuration Release --no-restore
dotnet test --no-restore --verbosity normal
```

 actions/starter-workflows

C# 

#### .NET Core Desktop

By GitHub Actions

Build, test, sign and publish a desktop application built on .NET Core.

[Set up this workflow](#)

```
dotnet test
msbuild $env:Solution_Name /t:Restore
/p:Configuration=$env:Configuration
$pfxcert_byte = [System.Convert]::FromBase64String("${{
secrets.Base64_Encoded_Pfx }}" )
```

 actions/starter-workflows

C# 

# GitHub Actionsを軽くおさらい

## Workflowの見かた

- on: Workflowをトリガーするイベント定義
- jobs: 実行単位(Job)を定義
- build: buildという名前のJobを定義
- steps: Jobで何をするかを定義

## ポイント

- stepsでActionやシェルを使い処理を連結

<> Edit new file

👁 Preview

```
1  name: .NET Core
2
3  on:
4    push:
5      branches: [ master ]
6    pull_request:
7      branches: [ master ]
8
9  jobs:
10    build:
11
12      runs-on: ubuntu-latest
13
14      steps:
15        - uses: actions/checkout@v2
16        - name: Setup .NET Core
17          uses: actions/setup-dotnet@v1
18          with:
19            dotnet-version: 3.1.301
20        - name: Install dependencies
21          run: dotnet restore
22        - name: Build
23          run: dotnet build --configuration Release --no-restore
24        - name: Test
25          run: dotnet test --no-restore --verbosity normal
26
```

細かい所を説明すると時間が足りないの  
で何を使えば何ができるかを紹介します

# PullRequestのマイルストーンチェック

やりたいこと:

PullRequestにマイルストーンを付ける運用にしているので忘れないようにしたい

アプローチ:

PullRequestにマイルストーンが付いていなかったらコメントする

# PullRequestのマイルストーンチェック

解決策:

octokit/request-actionを使ってGitHubのAPIを叩く

```
jobs:
  check:
    runs-on: ubuntu-latest
    steps:
      - uses: octokit/request-action@v2.x
        if: # ここでWebhook Payload の値を確認すれば良い
        with:
          route: POST /repos/:repository/pulls/:pull_number/reviews
          repository: ${ github.repository }
          pull_number: ${ github.event.pull_request.number }
          body: "PullRequest comment"
        env:
          GITHUB_TOKEN: ${ secrets.GITHUB_TOKEN }
```

# PullRequestのマイルストーンチェック

詳細なサンプルはこちらに公開しています

<https://github.com/MeilCli/actions/blob/master/.github/workflows/check-has-milestone.yml>

# ファイル生成の自動化

やりたいこと:

ツールで自動生成するファイルの生成をCIで自動化したい

アプローチ:

CIでファイルを自動生成し、差分があればPullRequestを作成する

# ファイル生成の自動化

解決策:

peter-evans/create-pull-requestを使う

```
jobs:
  check:
    runs-on: ubuntu-latest
    steps:
      - run: echo "example" > message.txt
      - uses: peter-evans/create-pull-request@v3
        with:
          commit-message: 'commit message'
          title: 'PR title'
          assignees: 'MeilCli'
          reviewers: 'MeilCli'
```



# Slackへ通知

やりたいこと:

CIのエラー通知や定期実行結果をSlackで見たい

アプローチ:

Slack Incoming Webhooksを使う

ref: <https://api.slack.com/messaging/webhooks>

# Slackへ通知

解決策:

8398a7/action-slackを使う & Webhook URLをGitHub ActionsのSecretに登録

```
jobs:
  notification:
    runs-on: ubuntu-latest
    steps:
      - uses: 8398a7/action-slack@v3
        with:
          status: custom
          custom_payload: |
            {
              text: 'Message from GitHub Actions'
            }
    env:
      SLACK_WEBHOOK_URL: ${ secrets.SLACK_WEBHOOK_URL }
```

# GitHub Actionsの高速化

やりたいこと:

ビルドに時間がかかるので早くしたい

アプローチ:

依存物やビルド結果などをキャッシュする

# GitHub Actionsの高速化

解決策:

actions/cacheを使う

```
jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
      - uses: actions/setup-java@v1
        with:
          java-version: 1.8
      - uses: actions/cache@v2
        with:
          path: ~/.gradle/caches
          key: ${ runner.os }-gradle-${ hashFiles('build.gradle') }
          restore-keys: |
            ${ runner.os }-gradle-
      - run: chmod +x gradlew
      - run: ./gradlew build
```

# まとめ

- クラシルでは以下のActionなどを活用しています
  - octokit/request-action
  - peter-evans/create-pull-request
  - 8398a7/action-slack
  - actions/cache
- GitHub Marketplaceで良さげなActionを探すといいです
- Actionがなければ自作することもできます