エッジAI における CI/CD - GitHub 編

自己紹介

岩永かづみ / IWANAGA Kazumi

- フリーランス, ZEN Architects 所属
- Microsoft MVP for Azure
- Azure における Infrastructure as Code や GitHub を用いた CI/CD 自動化が得意
- GitHub公認トレーナー

Twitter: @dz_
GitHub: @dzeyelid
Zenn: dzeyelid

CI/CD のおさらい

CI/CD のおさらい

- Continuous Integration/Continuous Delivery(Deployment)
- DevOps の考え方を取り入れるときにキーとなる自動化
- CI: 主に ビルド、テスト など
- CD: 主に デプロイ

GitHub における CI/CD

GitHub における CI/CD に関連したサービス

関連するサービス	説明
GitHub Actions	自動化の手順を記述したワークフローを実行できる
GitHub Marketplace	GitHub Actions で利用できるアクションを探すことができる
GitHub Packages	様々な言語に対応したパッケージレジストリ
GitHub Container Registry	コンテナイメージのレジストリ（GitHub Packages の一部）

GitHub Actions

GitHub Actions とは

ワークフローに処理を定義しておき、定義したトリガを契機に実行させる。

- アクション
- トリガ
- ランナー
- シークレット
- Environment

GitHub Actions のアクション

OSSで構成された処理のコンポーネント

- 自作
  - JavaScript で記述、または コンテナとして構成
  - 参考: カスタム アクションについて - GitHub Docs
- Marketplace
  - Verified creator や登録されたアクションを検索できる
- OSSで公開されているリポジトリ
- シェルスクリプトの実行
  - bash, PowerShell

GitHub Actions のトリガ

ワークフローを実行する契機を定義する

- プルリクエストを作成、編集、クローズされたとき
- プッシュされたとき
- issue が作成、更新、クローズされたとき
- REST API や手動による実行

くわしくは、

- 参考: Triggering a workflow - GitHub Docs
- 参考: Events that trigger workflows - GitHub Docs

GitHub Actions のランナー

- Github-hosted と Self-hosted
- GitHub-hosted
  - Ubuntu, Windows, macOS
  - 参考: GitHub ホステッド ランナーの概要 - GitHub Docs
- Self-hosted
  - 参考: About self-hosted runners - GitHub Docs

GitHub Actions のシークレット、Environment

機能	説明
シークレット	- 機密情報はシークレットに格納し、ワークフローから参照できる - ログ上でも目隠しされる - スコープは、リポジトリ、Environment、Oraganization 単位
Environment	承認者（reviewers）によるワークフロー実行の待機をさせたり、環境ごとのシークレットを設定できる

- 参考: Encrypted secrets - GitHub Docs
- 参考: デプロイに環境を使用する - GitHub Docs

おまけ: Branch Protection rule と組み合わせて強化

- Branch Protection rule（ブランチ保護ルール）の *Require status checks to pass before merging* を有効化することで、プルリクエストに対するワークフローの実行ステータスが成功した場合のみマージできるよう制限できる
- コードの品質保持やセキュリティ対策を行える
- 参考: About protected branches - GitHub Docs

GitHub Container Registry

GitHub Container Registry とは

- コンテナイメージを格納、配布できるレジストリ
- GitHub の認証を用いてアクセスできる

エッジAIにおける CI/CD の考察

エッジAIにおける CI/CD の考察

着目点	Webアプリケーションの場合	エッジAIの場合
デプロイするもの	アプリケーション	アプリケーション、**AIモデル**
デプロイ先	サーバー	**エッジ**

- 最近の Webアプリケーション開発では、クラウドプラットフォームの恩恵でデプロイが容易に
- エッジの場合、デプロイの仕組みを構成するのが大変？

エッジAIにおける CI/CD のデモ

エッジAIにおける CI/CD のデモ

- デプロイの仕組みに、Azure IoT Edge を利用
- AIモデルの生成は Azure Custom Vision を採用
- CI/CD のワークフローは **GitHub Actions** を利用
- IoT Edge で利用するコンテナイメージは **GitHub Container Registry** を利用

Azure IoT Edge について

- コンテナの技術を利用し、エッジのアプリケーション管理ができる
- アプリケーションのデプロイや状態の共有ができる

Azure Custom Vision について

- 任意の画像を学習し、分類やオブジェクトの検出ができる
- GUI による操作ができる
- 複数の形式でモデルのエクスポートができる
    - デモでは Dockerfile 形式でエクスポートを行う

デモシナリオ

1. Azure Custom Vision で学習する（イテレーションが作成される）
2. イテレーションIDを更新したプルリクエストを作成する
3. **CI** - Custom Vision モデルのエクスポート（Dockerfile）、ダウンロードする
    - 初回のモデルのダウンロード、一度ワークフローは失敗する（意図的、解説）
4. **CI** - Dockerfile（Custom Visionのモデル含む）をビルド、イメージをプッシュする
5. **CD** - IoT Edge で管理しているデバイスにデプロイする