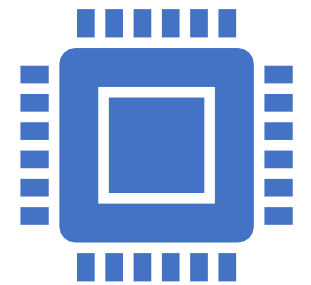


最近のモダンなCI/CD環境
～Jenkinsおじさんいらず
/GitHub Actions/etc～

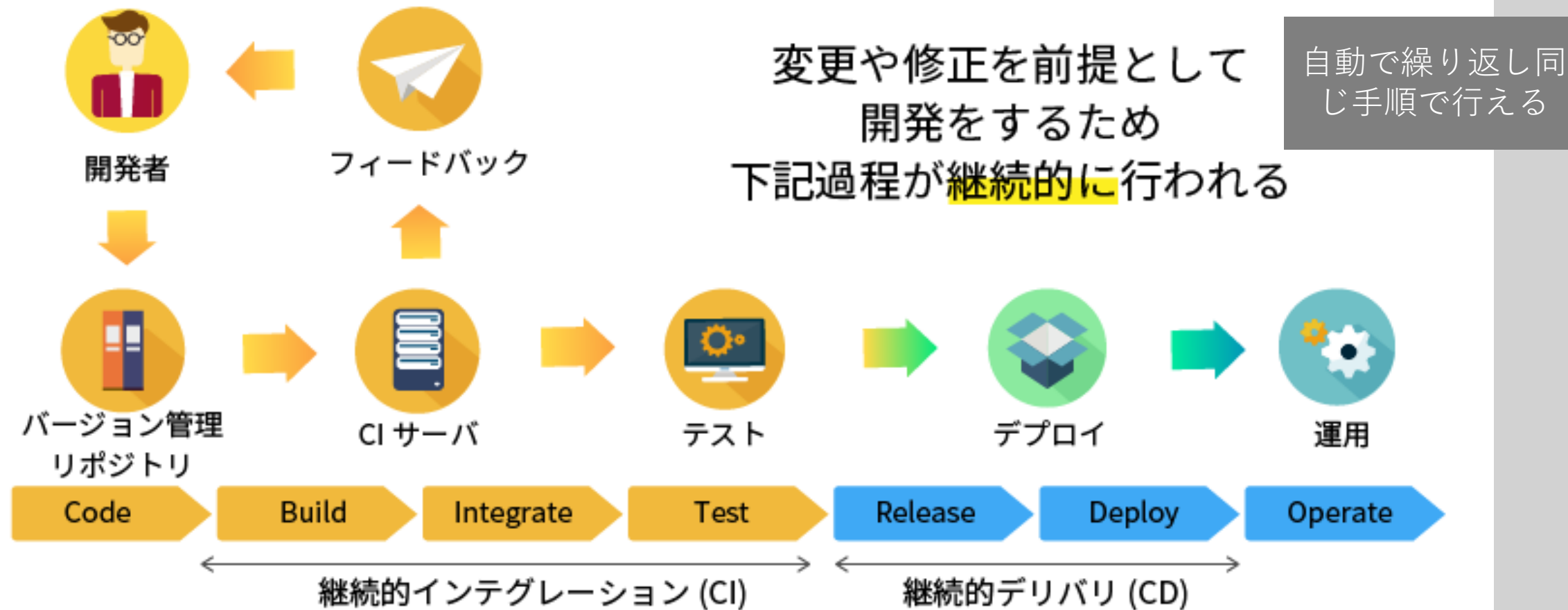


2021/3/17 荻原

はじめに

- 本日のテーマ
 - 「最近のモダンなCI/CD環境」と題して巷ではどんな感じのCI/CDツールが使われているかを紹介させていただき、なにがうれしいのか？それってうれしいのか？を（かなりの）私見を交えて説明させていただきます
- 本日の狙いは！
 - 一般的にモダンと言われているCI/CDとはどのようなもののかのイメージを掴んでもらう
 - GitHubを例に
 - モダンなCI/CDではどのようなことができるか
 - モダンなCICDはどんな感じで利用することができるのか

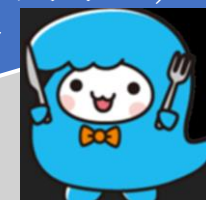
まずはチョットおさらいーCI/CDとは



で、モダンなCI/CDとは (オレオレ的な世の中の疎総論)

- **クラウドで**リソースを気にせずいつでも利用できる
 - ビルドやテストで必要に応じてAutoScalingしてくれる
 - コンテナとの親和性がよくいつもクリーンな状態でテストを実行してくれる (前のテスト結果などの副作用を気にせずに済む)
- **だれでも簡単に**パイプライン※の定義を行える
 - YAMLなどの簡易な記述形式でビルドやテストなどでやりたいことを宣言的に定義することができる
 - JenkinsなどCI/CDツール独自の設定や複雑なスクリプトの知識を必要とせずパイプラインを簡便に定義することができる (反対は詳しい人じゃないと設定やパイプラインの定義ができていないこと)
- **至れり尽くせり**で機能が付いている
 - 純粋なCI/CDの機能だけでなくその周辺の機能もテンコ盛りで付いてくる
 - CI/CDを起点として最初からそれらの周辺機能と連携できるようになっている


巷ではJenkinsに詳しく素敵なパイプラインの定義や実装ができる人を「Jenkinsおじさん」と言うこともある (元の意味は執事のキャラクター)







※:ビルド⇒テスト⇒(・・・)⇒デプロイなどのタスクを「いつ」「どんな順番で」「どのように」繋いでいくかを定義したもの



ちなみに . .

- モダンなものが良いか？モダンなものが自分たちにとってホントにうれしいか？は別
 - 最後に考察してみましょう！
- 

モダンなCI/CDの代表選手

製品	一口コメント
	<p>Travis CI</p> <ul style="list-style-type: none">• 昔からある定番のクラウドで利用できるCI• public（公開）利用であれば無料で利用可
	<p>Circle CI</p> <ul style="list-style-type: none">• みんな大好きCircle CI、人気あります• 便利な機能が満載
	<p>AWS CodeBuild AWS CodePipeline</p> <ul style="list-style-type: none">• Amazon Web Service好きならコレ• お値段も信頼と実績のAWS価格
	<p>GitHub Actions</p> <ul style="list-style-type: none">• 後発だが今モットも勢いがある!?• 後ほど紹介します

GitHub Actionsの前に ・ ・

そもそもGitHubってなによ！

GitHubとは

- ソースコード管理を行う **Gitリポジトリ**を中心に
- issue managementなど開発に必要な **多くの周辺機能**を備えた
- **クラウド**で
- public利用は**無料**で
- 利用することが出来る **素敵なクラウド環境**
- 主要な多くのOSSで利用されているのも特徴

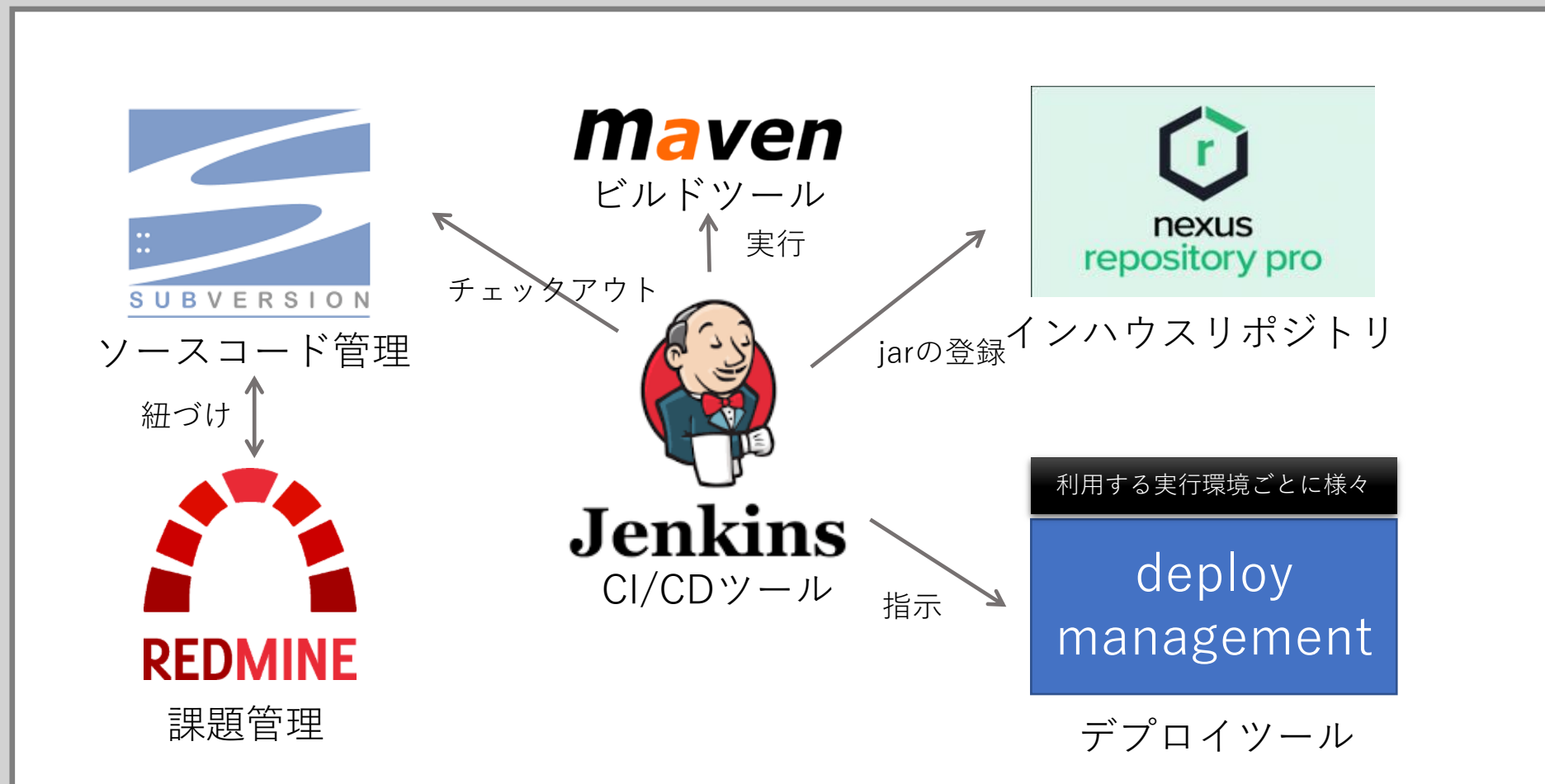
機能	一口説明
repository	ソースコード管理
pull request	ブランチのマージリクエスト
GitHub Issues	課題チケット管理
GitHub Packages	jarのリポジトリ
GitHub Pages	静的webページの公開
GitHub Projects	プロジェクト管理

豊富な周辺機能
(よく使われていると思うもの)

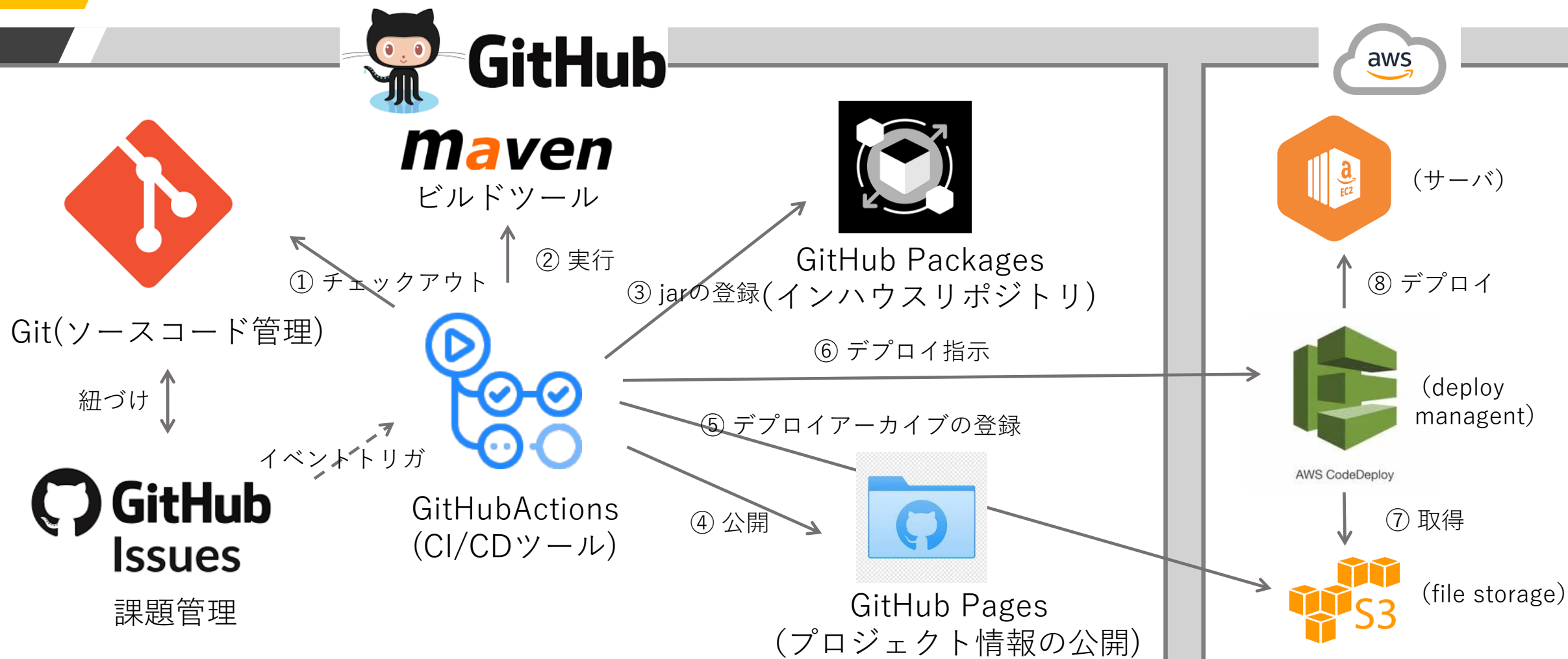
やっと本題 ～GitHubを使ったCI/CDの紹介～

その前にまずはよくある代表的なオンプレ環境

よくあるオンプレミスな環境



GitHubを使ったCI/CDの構成 (サンプルアプリの構成)



CI/CDで やっている こと

デモアプリ

- 前回のお題を実装したRESTAPI

GitHubActionsでやっていること

- Mavenを実行
- ビルド結果をCodeDeploy形式にアーカイブ
- アーカイブをAWS CLIでS3にアップ
- アップしたアーカイブをAWS CLIでCodeDeployに対してデプロイ指示

Mavenで実行

- ビルドしたjarをGitHub Packagesにアップ
- ビルドしたときの結果をmavenのreport機能で生成
- 生成したreportをmavenのsiteプラグイン機能でプロジェクト情報として生成
- 生成したコンテンツをmavenのpublish機能でGitHub Pagesにデプロイ

+ https://github.com/mamezou-tech/minna_de_kagaikatudou/tree/main/5th_cicd

実際に見
てみま
しょう！

- ポイント
 - JUnitとFindBugsのCIレポートは意図的にエラーがある状態にしてある
 - デプロイしたアプリからMicroProfile OpenAPI機能を使ってOpenAPIドキュメントを取得しSwaggerUIで表示
- ハンズオン & デモ
 - アプリを動かしてみる
 - SwaggerUIでREST APIのSpecを確認し実際にアプリを呼び出してみよう！
 - エラーを修正してCI/CDを実行
 - テストのエラーとFindBugsのエラーを解消しレポート結果を即座に確認
 - コードの割引率を変えデプロイ後のAPI挙動をSwaggerUIで試してみる
 - コンソール操作はせずにAWSのエセ本番環境に反映されていることを確認

GitHub Actionsの中身を見てみよう

StepByStepで見ていくけど雰囲気だけね！

コミットされたコードをビルドしてテストしてjarをデプロイするまでのシナリオ

まずはworkflowを動かすイベントを定義

```
on:  
  workflow_dispatch:  
    branches: [ main ]  
    paths:  
      - '5th_cicd/cicd-sample/**'  
  push:  
    branches: [ main ]  
    paths:  
      - '5th_cicd/cicd-sample/**'
```

ワークフローを実
行のボタンをポ
チっとされたら

該当のコードがコ
ミットされたら

- 他にも一杯組み込みのイベントトリガが用意されている
- それをYAMLで宣言するだけ！

次にビルド実行の定義

```
jobs:  
  build_and_test_and_deploy-jar:  
    runs-on: ubuntu-latest  
    steps:  
      - uses: actions/checkout@v2
```

Jobのお名前を定義

最新のubuntuで実行してね

チェックアウト

- Checkoutは用意されているActionを指定するだけ
- ActionとはMavenのプラグインみたいなもので、手順や操作をモジュール化し再利用可能にしたもの
- 実体はShellScript or TypeScript

次はJavaの設定

```
- name: Set up Java11
  uses: actions/setup-java@v1
  with:
    java-version: 11
```

java-versionで指定したJDKをセットアップしてね

えっ、Javaの設定ってなんで毎回するの？

- GitHub ActionsはJobごとにDockerコンテナを割り当てコンテナ上でJobを実行します
- なので、Jobが終わるとまっさらになります。
- よく言えば、副作用のないクリーンな環境で毎回テストを行える
- 悪く言えば設定や結果をローカルに保存しておくといったことができないので面倒

次はキャッシュ

```
- name: Cache Maven packages
  uses: actions/cache@v2
  with:
    path: ~/.m2/repository
    key: ${{ runner.os }}-m2-${{ hashFiles('**/pom.xml') }}
    restore-keys: ${{ runner.os }}-m2
```

要はチェックアウトしたpomのhash値をキーとしたキャッシュがあるが確認し、あったらそれを使う

えっ、キャッシュってなん
でー？

- Mavenはローカルにない依存ライブラリをリポジトリから取ってきますが、Jobごとにこれも消えてしまう
- 毎回外部のリポジトリから依存ライブラリを取得すると時間が掛かるので、pomが変わってない場合はキャッシュしたものを使うようにしている

やっとビルドとテストとjarのデプロイ

```
- name: Build & Test & Deploy
  run: |
    cd 5th_cicd/cicd-sample
    mvn -B clean deploy --file pom.xml
  env:
    GITHUB_TOKEN: ${github.token} # for deploy to GitHub Packages.
```

これはmavenをそのまま使います

えっ、使うってどうやるの？

- runタスクを使うとOSのshellコマンドがそのまま使えます
 - が、しかし1行ずつのコマンドライン実行なのでIF文やFOR文などの制御文は使えません
 - ⇒なので基本的にワンライナー、そしてシェル芸人へ
 - ⇒ただ、シェルスクリプトは普通に実行できるのでIF文などの制御文が使いたい場合はシェルスクリプトにすればよいが（粒々できるのがイヤ）
- GITHUB_TOKENってなに？
 - ビルドしてできたjarはGitHub Packagesに登録しますが。uploadするときにアクセス権が必要となります。そのアクセストークンの設定

EC2へデプロイするためのアーカイブの作成

```
- name: prepare deployment
  run: |
    cd 5th_cicd/cicd-sample
    mkdir cicdSample
    mv ./target/libs ./cicdSample
    mv ./target/cicd-sample-main.jar ./cicdSample
    mv ./deployment/appspec.yml ./cicdSample
    mv ./deployment/scripts ./cicdSample
```

泥臭くランラン♪します

- と、ここまですり雰囲気は分かっていただけたと思うので、以降は興味があったら愛でてみてくださいmm

GitHub Actionsを評価してみる

ここが凄いぞGitHub Actions

- 第1位は

- タダ！いくらCPUを使ってもタダ！

ただしpublic利用に限る。また一部例外あり

- 第2位は


- マトリックステスト（これをオンプレでやるのは実質的に無理。特にOSのマトリックステストとか）
 - MatrixTests.ymlを見てみよう！

- 第3位は

- リポジトリに対するイベント以外にもイベントトリガーにすることができる
 - 例えば・・・issueに対する操作など

MSさまの社会貢献とあってありがたく使わせていただきます！

だけどチ ョ ッ ところが残念GitHubActions

- GitHubActionsにはJenkinsのようなDashboardがない！なので結果が分かりづらい
 - えっ、バッジで分かるって？
- GitHubActionsには他のWorkflowへのイベント通知機能がないのでWorkflow間の連携ができない（なんとなくJenkinsはできたような気が・・・教えてJenkinsおじさん）
- GitHub Issuesで管理できるステータスはOpenとCloseだけでカスタムはできない。またカスタムフィールドも作れない
 - あっ、これはGitHubActionsと関係ないネ
- GitHub Pagesのドキュメントルートはリポジトリの1か所だけしかマッピングできない。なので異なるリポジトリパスの1部分を公開することができない
 - あっ、これもGitHubActionsと関係ないネ

まとめ

確かなこと

- モダンな方はオールインワンで連携機能ごとの設定が不要でいろいろなものをすぐ使うことができる
- 一方のオンプレは色々構築&設定する手間はあるが、それぞれの機能に特化したものを単品ごとに使うので出来ることは多い
- なので、結局は・・・

すぐ使えるが機能はちょっと物足りない vs
手間がかかるけど機能は豊富のトレードオフ

モダンの幻想？

● **クラウド**でリソースを気にせずいつでも利用できる

- これはそのとおり！！🙌

? **だれでも簡単に**パイプラインの定義を行える

- YAMLとpomを見てみると・・・

? **至れり尽くせり**で機能が付いている

- そうとも言えるが単品モノと比較すると・・・

Jenkinsおじさんは不要になるが、それとは別にpom職人やシェル芸人、YAML番長は必要でモダンだからと言って難しいことや複雑なことを簡単に出来る訳ではない！（敢えて言うなら少しシンプル程度・・・）

ご静聴ありがとうございました