

# 未来を見据えた CI/CD

～10年後も使える ビルド・テストパイプライン～

*Shota Nonaka  
CASAREAL,inc.*

Nov. 27, 2022

# 自分について



株式会社カサレアル

プロフェッショナルソリューション技術部 Cloud Native Technical Officer

野中 翔太  
*Nonaka Shota*



# △CASAREAL 株式会社カサレアル

- MISSION -

- SERVICE -

# △CASAREAL 株式会社カサレアル

## - MISSION -

世の中に貢献できる多くの技術者と事業者を育て  
世の中を支えるシステムを構築していく

## - SERVICE -

# △CASAREAL 株式会社カサレアル

## - MISSION -

世の中に貢献できる多くの技術者と事業者を育て  
世の中を支えるシステムを構築していく

## - SERVICE -

3つの事業をワンストップで提供



# クラウドネイティブ技術支援サービス

## クラウドネイティブ技術支援サービス

- クラウドインフラ環境構築支援

## クラウドネイティブ技術支援サービス

- クラウドインフラ環境構築支援
- パイプライン自動化支援

## クラウドネイティブ技術支援サービス

- クラウドインフラ環境構築支援
- パイプライン自動化支援
- オンプレ移行マイクロサービス設計支援

## クラウドネイティブ技術支援サービス

---

- クラウドインフラ環境構築支援
- パイプライン自動化支援
- オンプレ移行マイクロサービス設計支援
- クラウドネイティブ道場（トレーニングコース）の提供

# CI/CD って？

ソフトウェア開発におけるビルド・テストやデプロイを自動化し、これらを継続的に行う仕組み。

用語	意味	主なタスク例
CI	継続的インテグレーション	JUnit テスト Gradle ビルド Docker イメージビルド & プッシュ
CD	継続的デリバリー	デプロイ etc...

# CI/CD って？

ソフトウェア開発におけるビルド・テストやデプロイを自動化し、これらを継続的に行う仕組み。

用語 意味

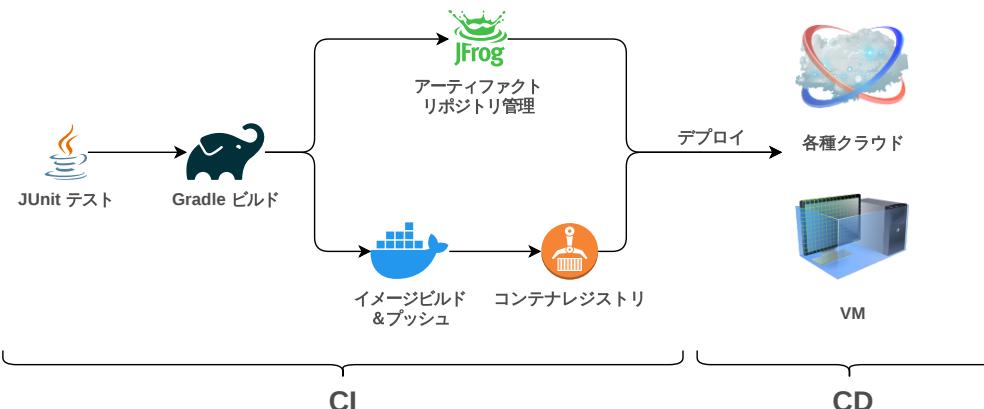
主なタスク例

CI 継続的インテグレーション

JUnit テスト  
Gradle ビルド  
Docker イメージビルド & プッシュ

CD 継続的デリバリー

デプロイ etc...



# CI/CD は必要？

ビジネスの価値をより確実かつ迅速にエンドユーザーに届けるために必要です。

## CI/CD の役割

# CI/CD は必要？

ビジネスの価値をより確実かつ迅速にエンドユーザーに届けるために必要です。

## CI/CD の役割

- ビルド・テスト・デプロイの高速化（自動化）
- 自動化に伴う人的エラーの防止

# CI/CD は必要？

ビジネスの価値をより確実かつ迅速にエンドユーザーに届けるために必要です。

## CI/CD の役割

- ビルド・テスト・デプロイの高速化（自動化）
- 自動化に伴う人的エラーの防止

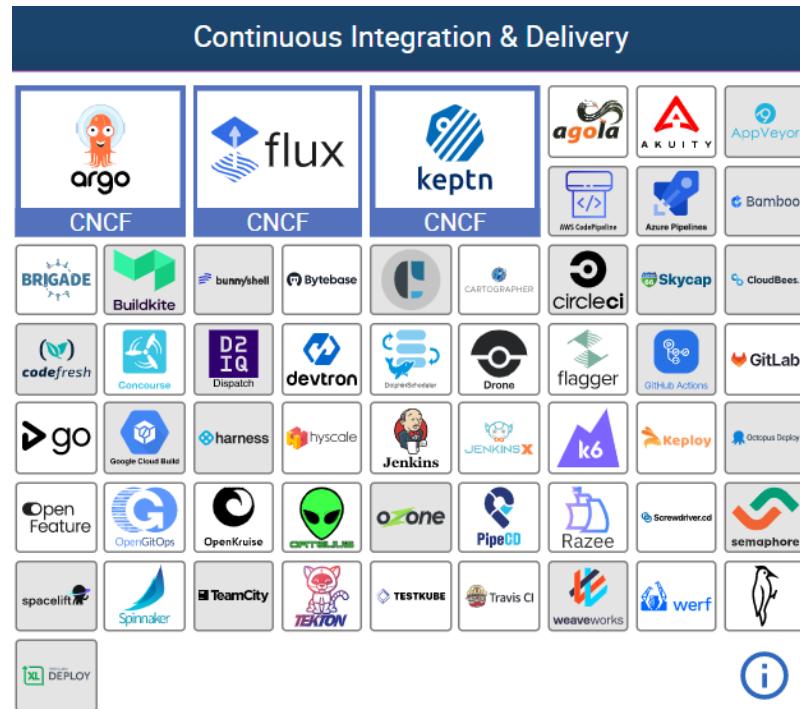
リリースサイクルを高速化することで、ビジネスの価値をより確実かつ迅速にエンドユーザーに届けることができます。

# CI/CD を実現するツール

- CNCF Cloud Native Interactive Landscape (<https://landscape.cncf.io/>)

# CI/CD を実現するツール

- CNCF Cloud Native Interactive Landscape (<https://landscape.cncf.io/>)



# 何を選ぶのがベスト？

もちろん、ご自身のプロダクトに合ったツールを選択するのは前提ですが、そもそも…

# 何を選ぶのがベスト？

もちろん、ご自身のプロダクトに合ったツールを選択するのは前提ですが、そもそも…

- ベストの順位は時間経過（ユーザのニーズやビジネスモデルの変化）によって変動しませんか？

# 何を選ぶのがベスト？

もちろん、ご自身のプロダクトに合ったツールを選択するのは前提ですが、そもそも…

- ベストの順位は時間経過（ユーザのニーズやビジネスモデルの変化）によって変動しませんか？
- 何を選択したとしても、ツールに依存しないようにパイプラインを作成しておくことが重要ではないでしょうか？

# 何を選ぶのがベスト？

もちろん、ご自身のプロダクトに合ったツールを選択するのは前提ですが、そもそも…

- ベストの順位は時間経過（ユーザのニーズやビジネスモデルの変化）によって変動しませんか？
- 何を選択したとしても、ツールに依存しないようにパイプラインを作成しておくことが重要ではないでしょうか？
- 何を選択したとしても、我々はプロダクトの成長を止めないように全力を尽くすべきではないでしょうか？

# 今、求められていることは？

様々なツールの選択肢でありふれている今の時代、  
10年後も廃れないプロダクトであり続けるためには、

が重要ではないでしょうか。

# 今、求められていることは？

様々なツールの選択肢でありふれている今の時代、  
10年後も廃れないプロダクトであり続けるためには、

- 数多くの選択肢に惑わされないこと

が重要ではないでしょうか。

# 今、求められていることは？

様々なツールの選択肢でありふれている今の時代、  
10年後も廃れないプロダクトであり続けるためには、

- 数多くの選択肢に惑わされないこと
- ビルド・テスト・デプロイといった普遍的なフローを再利用可能な資産にしておくこと

が重要ではないでしょうか。

# 今、求められていることは？

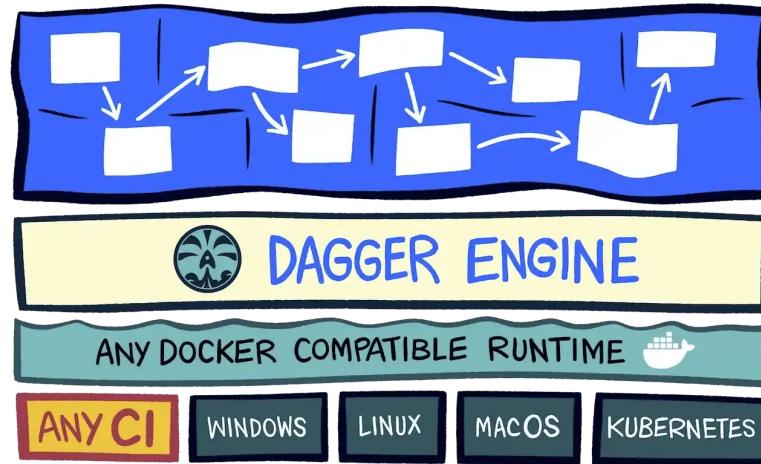
様々なツールの選択肢でありふれている今の時代、  
10年後も廃れないプロダクトであり続けるためには、

- 数多くの選択肢に惑わされないこと
- ビルド・テスト・デプロイといった普遍的なフローを再利用可能な資産にしておくこと

が重要ではないでしょうか。

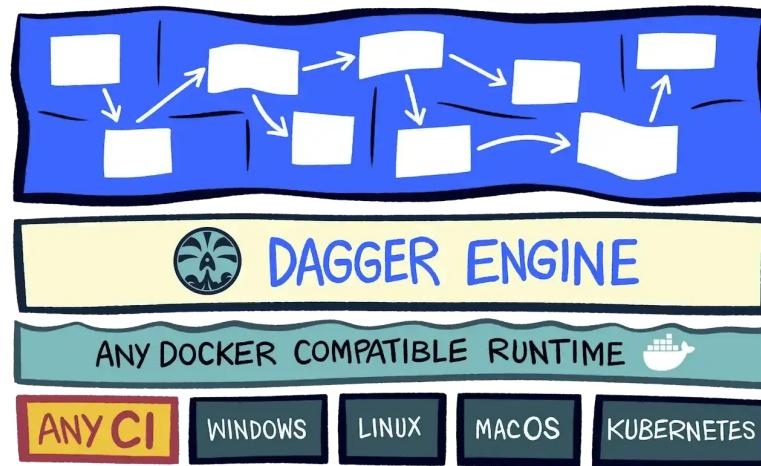
**Dagger** はその手助けをしてくれます。

# Dagger とは



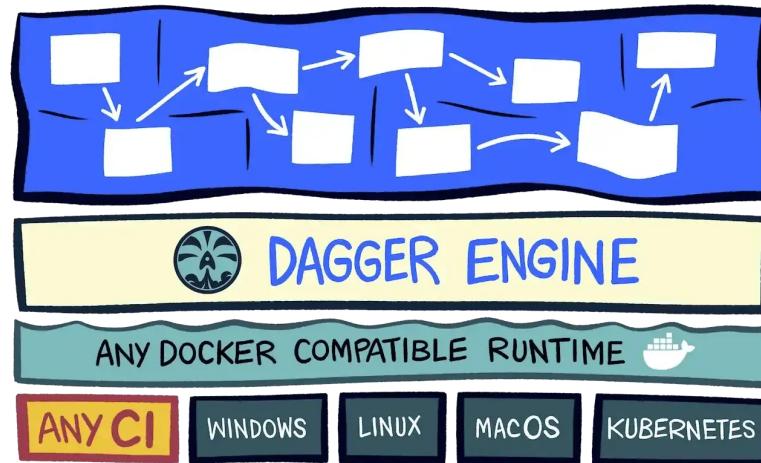
# Dagger とは

- ポータブルな CI/CD 開発キットというコンセプトの OSS(Apache License 2.0)



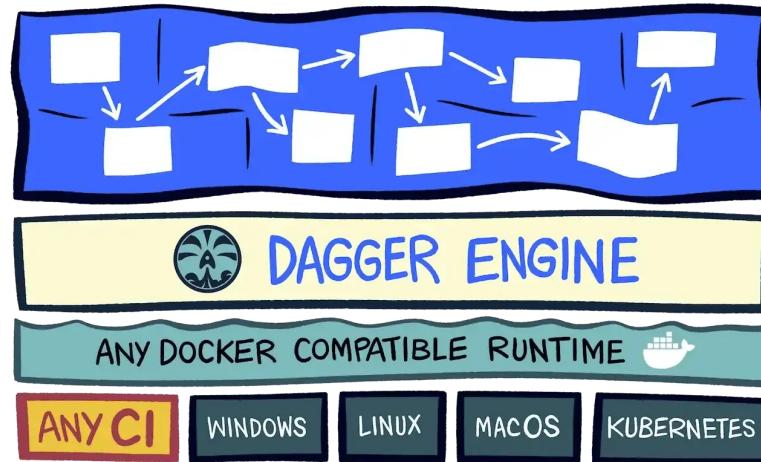
# Dagger とは

- ポータブルな CI/CD 開発キットというコンセプトの OSS(Apache License 2.0)
- Docker の生みの親 (`Solomon Hykes`) がプロジェクトを推進している



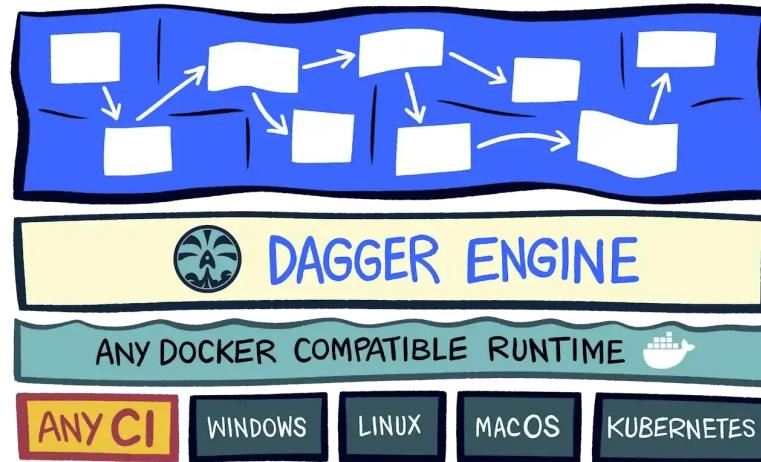
# Dagger とは

- ポータブルな CI/CD 開発キットというコンセプトの OSS(Apache License 2.0)
- Docker の生みの親 (`Solomon Hykes`) がプロジェクトを推進している
- 一度パイプラインを作成すれば、どこでも（ローカルでも）実行できる

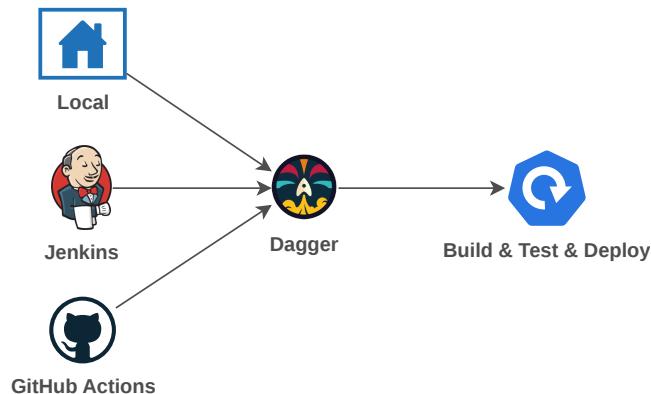


# Dagger とは

- ポータブルな CI/CD 開発キットというコンセプトの OSS(Apache License 2.0)
- Docker の生みの親 (`Solomon Hykes`) がプロジェクトを推進している
- 一度パイプラインを作成すれば、どこでも（ローカルでも）実行できる
- パイプラインは CUE 言語で記述する

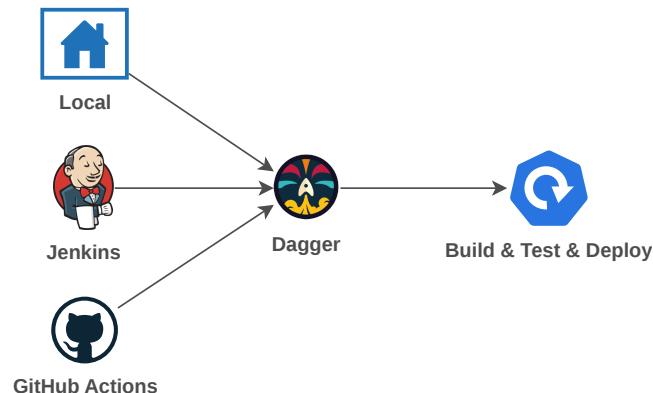


# Dagger は何を解決するか



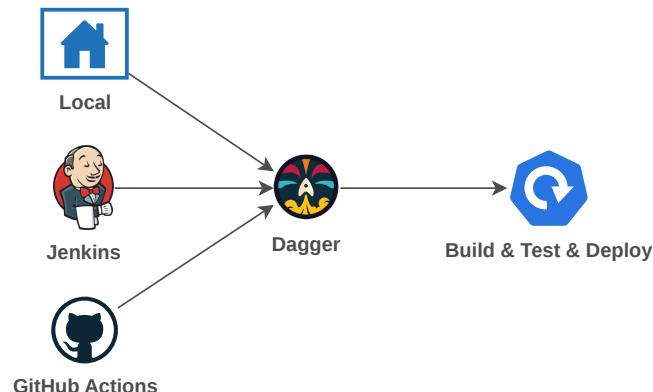
# Dagger は何を解決するか

- 様々な CI/CD ツールにロックインしない
  - ツールを Dagger 置き換えるのではなく、普遍的なフローを再利用可能にする



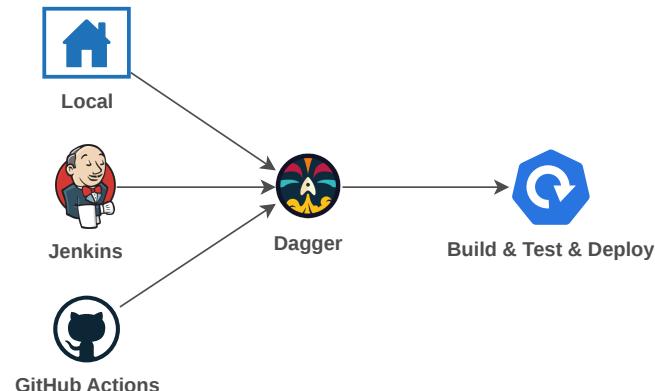
# Dagger は何を解決するか

- 様々な CI/CD ツールにロックインしない
  - ツールを Dagger 置き換えるのではなく、普遍的なフローを再利用可能にする
- ローカルで CI/CD を実行できる
  - 記述したパイプラインの動作確認が容易



# Dagger は何を解決するか

- 様々な CI/CD ツールにロックインしない
  - ツールを Dagger 置き換えるのではなく、普遍的なフローを再利用可能にする
- ローカルで CI/CD を実行できる
  - 記述したパイプラインの動作確認が容易
- 既存の Dockerfile の記述をそのまま利用することもできる（互換性）
  - 今ある資産も無駄にならない



# Dagger を使ってみる

いくつかデモンストレーションを実施します。

1. Dagger で Gradle ビルドしてみる
2. Dagger で Docker ビルドしてみる
3. Dagger で Docker イメージをロードしてみる
4. Dagger で Docker プッシュしてみる
5. GitHub Actions で Dagger を実行してみる

# デモンストレーションの実施環境

## バージョン 補足

Ubuntu (ホスト OS)	22.04	WSL 上の Docker コンテナ
Dagger	v0.2.36	
Gradle	7.5.1	ビルド環境として DockerHub の `gradle:7.5.1-jdk17` を使用
Corretto (OpenJDK のディストリビューション)	17	Java 実行環境として DockerHub の `amazoncorretto:17` を使用
ECR	-	AWS のマネージドなコンテナレジストリサービス
AWS App Runner	-	AWS のフルマネージドなコンテナデプロイサービス

# デモンストレーションのディレクトリ構成

主に、`dagger.cue` を編集する。

```
├── build.gradle
├── dagger.cue // パイプラインを記述するファイル
├── pom.xml
└── settings.gradle
└── src
    ├── main
    │   └── java
    │       └── com
    │           └── example
    │               └── springboot
    │                   ├── Application.java
    │                   └── HelloController.java
    └── test
        └── java
            └── com
                └── example
                    └── springboot
                        └── HelloControllerTest.java
```

# デモンストレーションのディレクトリ構成

主に、`dagger.cue` を編集する。

```
├── build.gradle
├── dagger.cue // パイプラインを記述するファイル
├── pom.xml
└── settings.gradle
└── src
    ├── main
    │   └── java
    │       └── com
    │           └── example
    │               └── springboot
    │                   ├── Application.java
    │                   └── HelloController.java
    └── test
        └── java
            └── com
                └── example
                    └── springboot
                        └── HelloControllerTest.java
```

# デモンストレーションのアプリ

`/`に HTTP リクエストを送信すると、`Hello Dagger!` という文字列を返す簡単な Web アプリケーション。

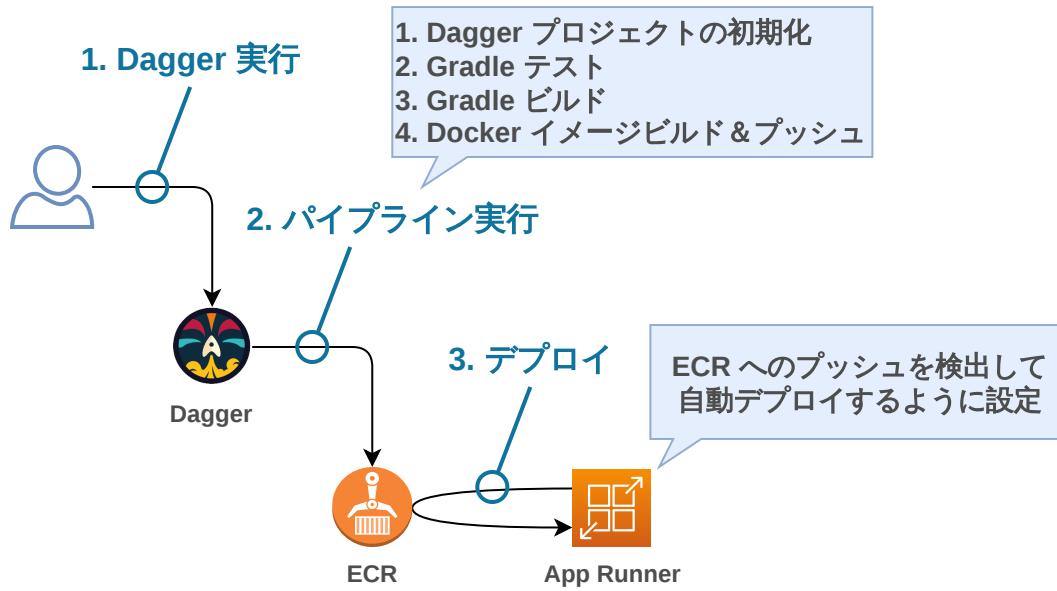
```
package com.example.springboot;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class HelloController {

    @GetMapping("/")
    public String index() {
        return "Hello Dagger!";
    }
}
```

# デモンストレーションのイメージ（ローカル）



# Dagger パイプラインの基本構成

大まかに以下の 4 つの項目から構成されます。

- パッケージ
- インポート
- クライアント
- アクション

# Dagger パイプラインの基本構成

- パッケージ

任意のパッケージ名を指定します。 (未指定だと Dagger 実行時にエラーになります)

```
package sample
```

- インポート

使用するパッケージを記載します。 (未使用のパッケージがあると Dagger 実行時にエラーになります)

```
import (
    "dagger.io/dagger"
    ...
)
```

# Dagger パイプラインの基本構成

- クライアント

ホストのファイルシステムに関する設定等を記載します。

下記の例では、カレントディレクトリの読み込みと、アクションの gradle\_build の出力を `build` ディレクトリへ書き込む設定をしています。

```
client: {
    filesystem: {
        "./": read: contents: dagger.#FS
        "./build": write: contents: actions.gradle_build.contents.output
    }
}
```

# Dagger パイプラインの基本構成

- アクション

`dagger do アクション`で実行するアクションの定義を記載します。  
下記の例では、gradle\_build というアクションを定義しています。

```
dagger.#Plan & {
    actions: {
        gradle_build: #GradleBuild & {
            ...
        }
    }
}
```

# Dagger で Gradle ビルドしてみる

Dagger で Gradle ビルドを実行し、jar が作成されることを確認します。

# Dagger で Docker ビルドしてみる

Dagger で Docker ビルドを実行し、パイプラインが正常に終了することを確認します。  
また、ローカルに Docker イメージが作成されていないことも確認します。

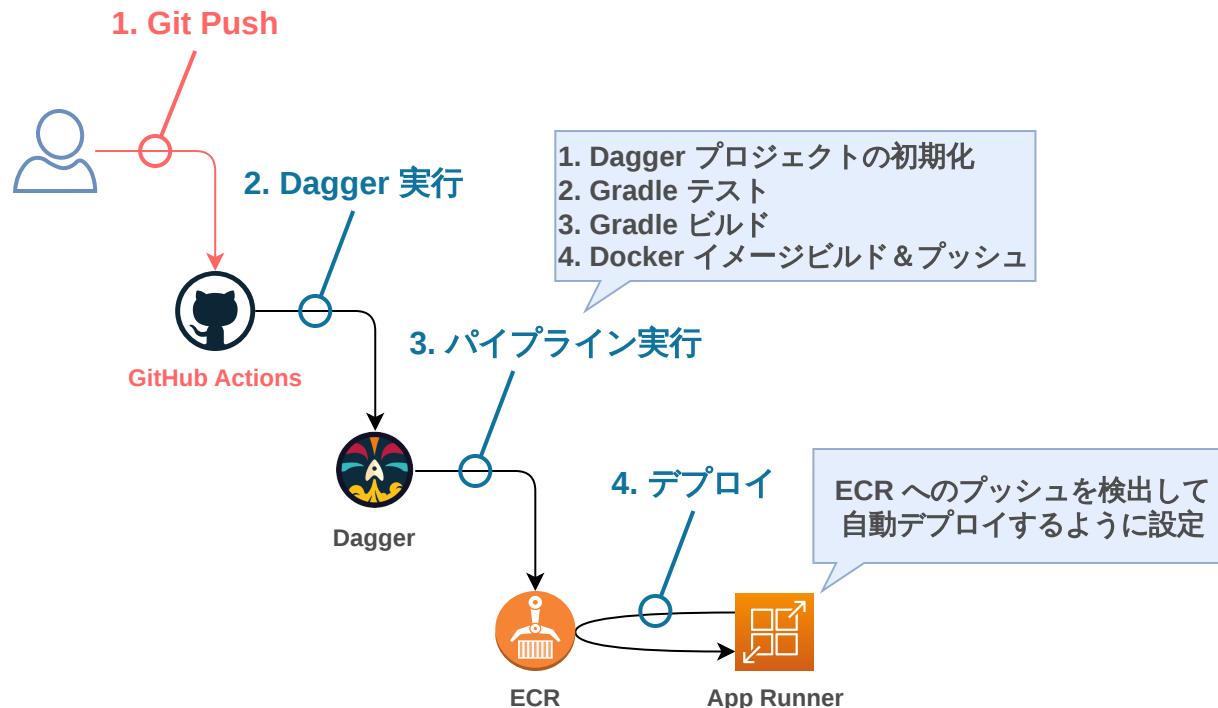
# Dagger で Docker イメージをロードしてみる

Dagger で Docker ビルドを行うと、BuildKit 上でイメージが作成され、ローカルには作成されません。しかし、開発中に正しくイメージが作成されていることを確認したい場合もあります。その際は、ビルドしたイメージをローカルにロードすることができます。

# Dagger で Docker プッシュしてみる

# デモンストレーションのイメージ (GitHub 連携)

ローカルとの違いは Dagger の呼び出し元の GitHub Actions のみ。  
パイプラインを再利用。



# ローカル

```
dagger project init  
dagger project update  
dagger do image_push
```

# GitHub Actions

```
jobs:  
  ...  
  
dagger:  
  - name: Image Push  
    uses: dagger/dagger-for-github@v3  
    with:  
      version: 0.2  
      cmds: |  
        project init  
        project update  
        do image_push
```

# ローカル

```
dagger project init  
dagger project update  
dagger do image_push
```

# GitHub Actions

```
jobs:  
  ...  
  dagger:  
    - name: Image Push  
      uses: dagger/dagger-for-github@v3  
      with:  
        version: 0.2  
      cmds: |  
        project init  
        project update  
        do image_push
```

# ローカル

```
dagger project init  
dagger project update  
dagger do image_push
```

# GitHub Actions

```
jobs:  
  ...  
  dagger:  
    - name: Image Push  
      uses: dagger/dagger-for-github@v3  
      with:  
        version: 0.2  
      cmds: |  
        project init  
        project update  
        do image_push
```

# GitHub Actions で Dagger を実行してみる

## コードの詳細について



Repository [casa-snona/dagger-demo](https://github.com/casa-snona/dagger-demo)

# Dagger の導入事例



「テクマトリックス NEO」の開発で Dagger を利用しています。

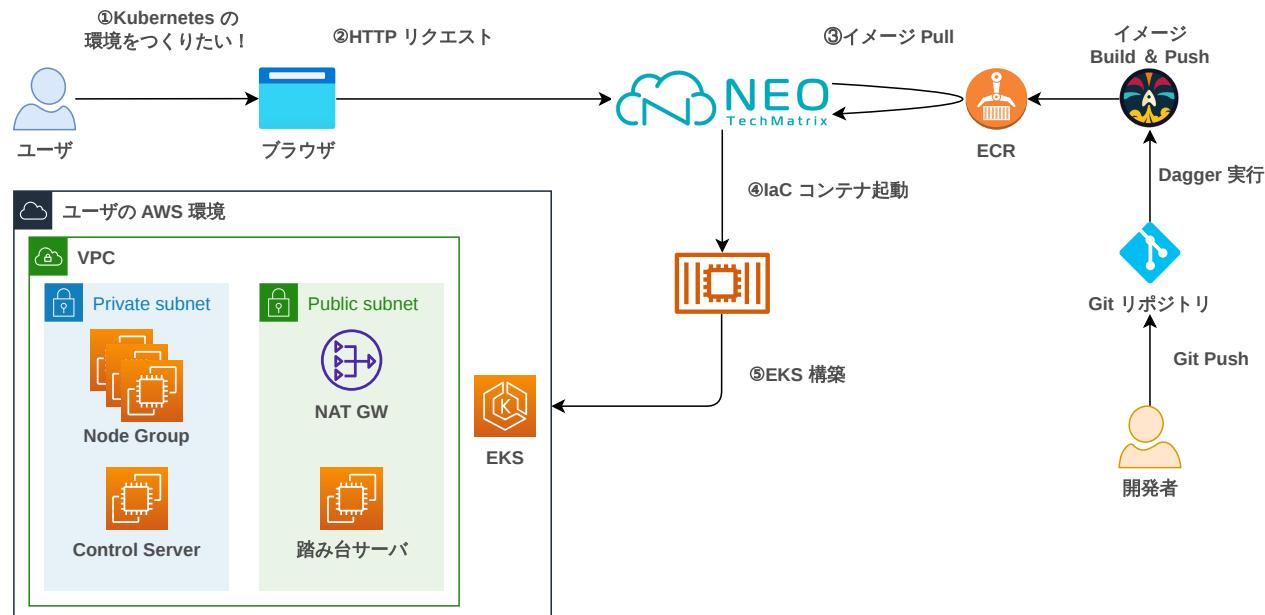
NEOは、日本にクラウドネイティブのテクノロジーをより早くより多く広めていくためのサービスです。クラウドネイティブ導入の伴走者として、初期教育から導入、運用まで幅広く支援をします。



# NEO のアーキテクチャと Dagger の役割

NEO は、簡単なブラウザ操作でネットワークや各種 OSS をインストールした VM の構築など、開発環境に必要なクラウドインフラを即座に提供します。

クラウドインフラを構築するための IaC を実行するコンテナイメージを Dagger で作成しています。



# 最後に...

様々なツールの選択肢でありふれている今の時代、

10年後も廃れないプロダクトであり続けるためには、数多くの選択肢に惑わされず、

ビルド・テスト・デプロイといった普遍的なフローを再利用可能な資産にしておくこと

が重要だと思います。

その手助けをしてくれる Dagger をぜひお勧めします。

