

# 2025年 夏季インターンシップ

GitHub Actions ハンズオン

テクノロジーオフィス リンウェイトン



SAKURA internet

# 目次

- 進み方説明
- GitHub Actions 入門
- GitHub Actions の初期設定
- Composite Actions の運用 SKIP
- sacloud\_apprun\_actions を使った AppRun へのデプロイ

# 進み方説明



SAKURA internet

# 進み方説明

- まずは、下記のハンズオン用のリポジトリをフォークしよう

```
https://github.com/ippnpeople/actions-hands-on
```

- このパートは、フォークしたリポジトリの `marp/slides/slide.pdf` で説明
- 具体的な手順は下記のREADMEに記載されているので、それぞれのREADMEを参照しながら進めていこう
  - `00_introduction_Github Actions/README.md`
  - `01_fork_and_setup/README.md`
  - `02_composite_actions/README.md` SKIP
  - `03_sacloud_apprun_actions/README.md`
- 目的: GitHub Actions の基礎を理解し、Actions を使ったアプリケーションのデプロイフローを体験すること

# GitHub Actions 入門

00\_introduction\_Github.Actions

# GitHub Actions とは？

バージョン管理システムであるGitHub上に提供する、ワークフローを自動化するCI/CDツール

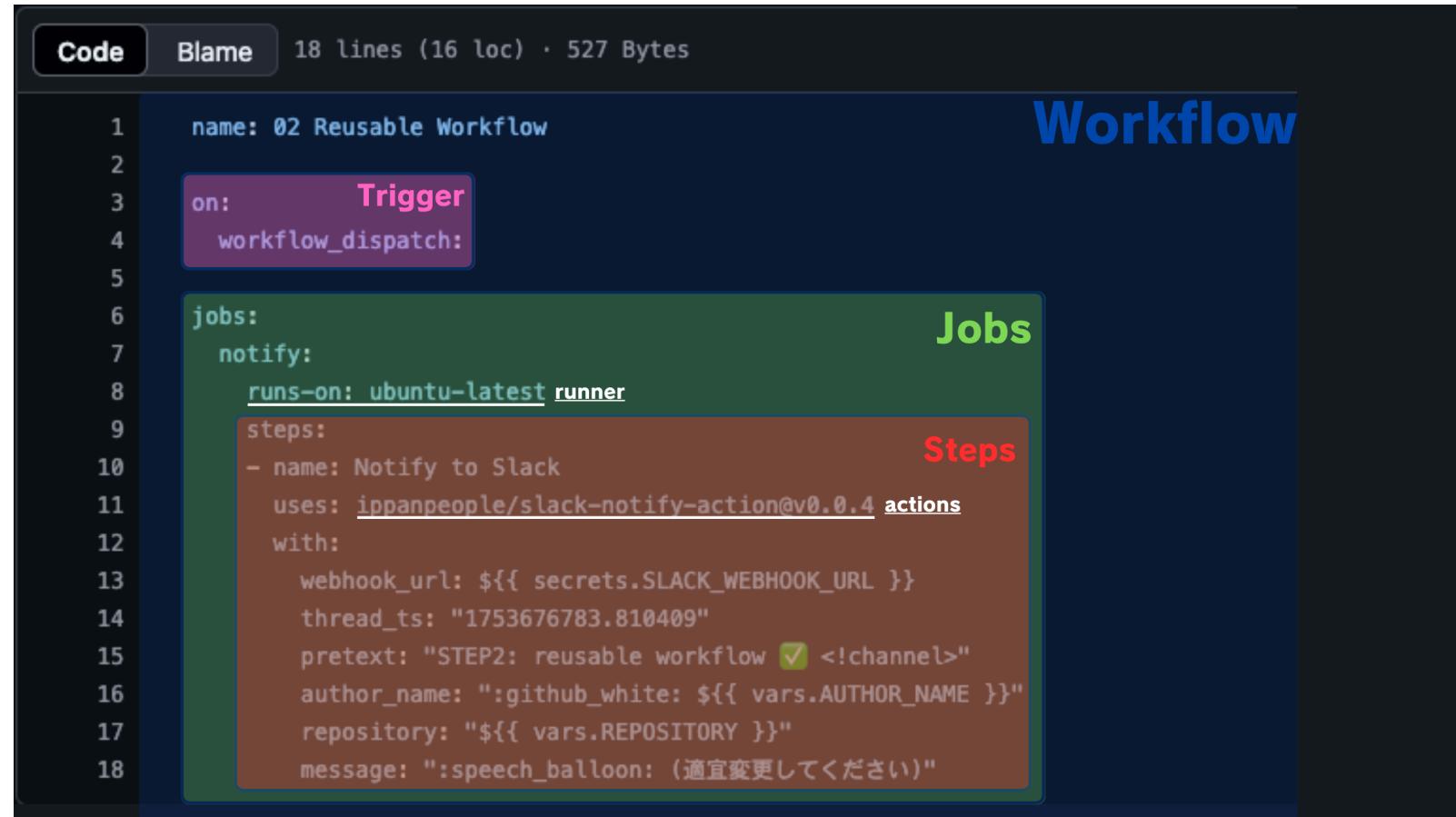
- できごと
  - イベント駆動型で、変更されたコードに応じて自動的に特定の処理・ビルド・テスト・デプロイを実行
- 動作するところ
  - ワークフローの定義に基づき、GitHub上やセルフホストされたランナーで動作
- ユーザビリティ
  - GitHubのUI上で簡単に設定・管理が可能、CLIやAPIからも操作可能

# GitHub Actions の基本コンポーネント

- **Actions**
  - 自動化ツールのこと。actions が小文字の際にワークフロー内で特定のタスクを実行する再利用可能なコード単位(プラグインのようなもの)
- **Workflow**
  - `.github/workflows/*.yml` で定義される自動化プロセス
- **Trigger**
  - ワークフローを開始するイベント(例: `push`、`pull_request`、手動実行など)
- **Jobs**
  - 同じランナー環境(仮想マシンなど)で実行されるステップの集合。並列または順次実行が可能

# GitHub Actions の基本コンポーネント

- Steps
  - ジョブ内の個々のタスク。コマンド実行やアクションの呼び出しができる



```
Code Blame 18 lines (16 loc) · 527 Bytes
```

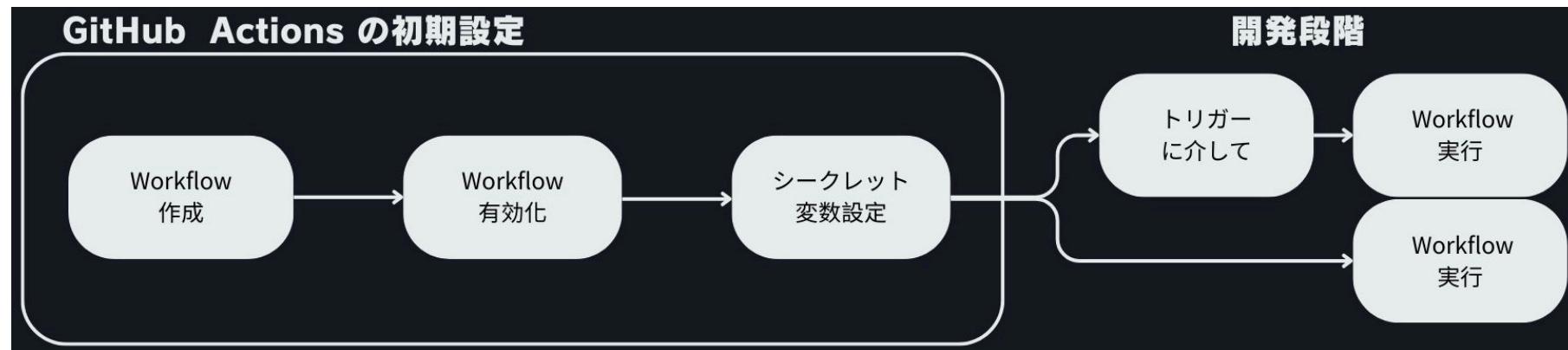
```
name: 02 Reusable Workflow
on: Trigger
  workflow_dispatch:

jobs:
  notify:
    runs-on: ubuntu-latest
    steps:
      - name: Notify to Slack
        uses: ippanpeople/slack-notify-action@v0.0.4 actions
        with:
          webhook_url: ${{ secrets.SLACK_WEBHOOK_URL }}
          thread_ts: "1753676783.810409"
          pretext: "STEP2: reusable workflow ✅ <!channel>"
          author_name: ":github_white: ${ vars.AUTHOR_NAME }"
          repository: "${ vars.REPOSITORY }"
          message: ":speech_balloon: (適宜変更してください)"
```

The screenshot shows a GitHub Actions workflow configuration. The code is organized into three nested scopes: **Workflow**, **Jobs**, and **Steps**. The **Workflow** scope contains a job named "notify" that runs on an "ubuntu-latest" runner. This job contains a single step that uses the "ippanpeople/slack-notify-action" action version v0.0.4. The step's configuration includes variables for the webhook URL, thread timestamp, pretext, author name, repository, and message.

# GitHub Actions の利用方法

1. Workflow ファイルを作成
2. Workflow を有効化
3. シークレットと変数の設定
4. Workflow を実行



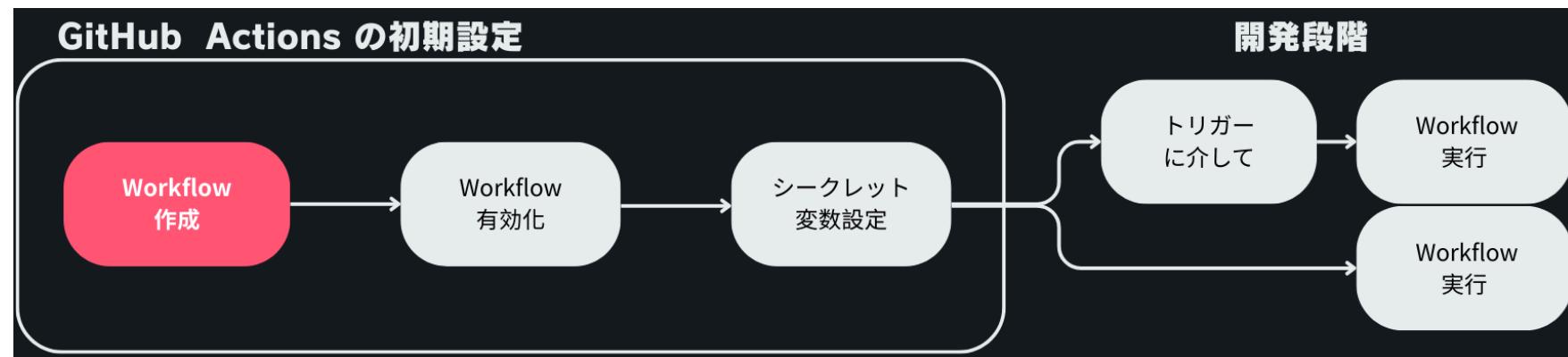
# GitHub Actions の初期設定

## 01\_fork\_and\_setup

# You Need a Workflow

Github Actions を利用するため、自動化プロセスであるワークフローを定義する必要がある

- ・ 内容: 基本的に先程紹介した基本コンポーネントを組み合わせて作成
- ・ タスクの実行方法:
  - **run:** シェルコマンドを実行する <- 01\_fork\_and\_setup.yml
  - **uses:** 他のアクションを呼び出す <- 03\_sacloud\_apprun\_actions.yml



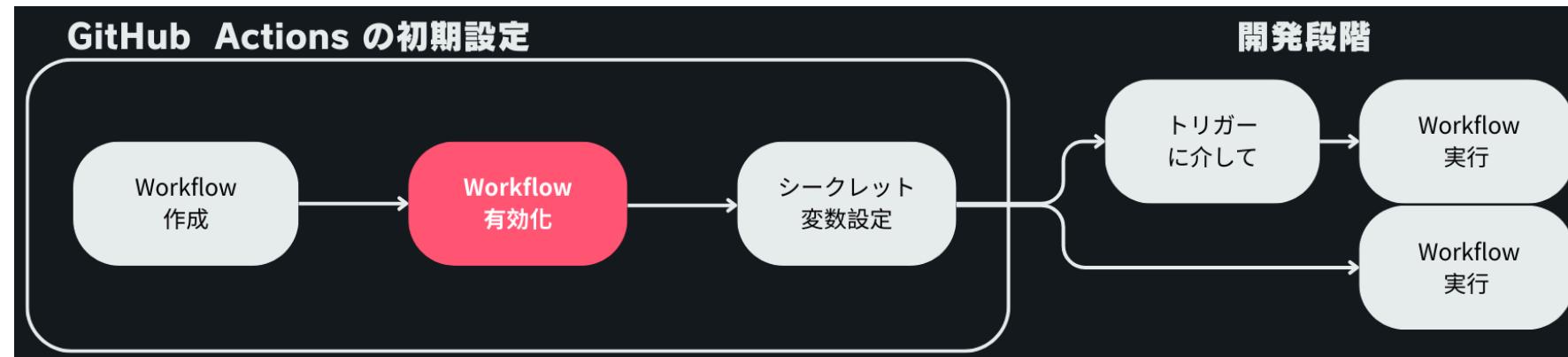
# You Need a Workflow

`01_fork_and_setup.yml` を例として、見てみよう:

- 用途: ハンズオンの進捗を確認するため、各ハンズオンが終わったたびに、Slack に通知し、進捗を報告用スレッドから確認できるように
- トリガー: 手動実行
- 実行すること
  - json payloadを作成
  - curl で payload を Slack Webhook に送信
  - エラーハンドリング

# Workflow の有効化

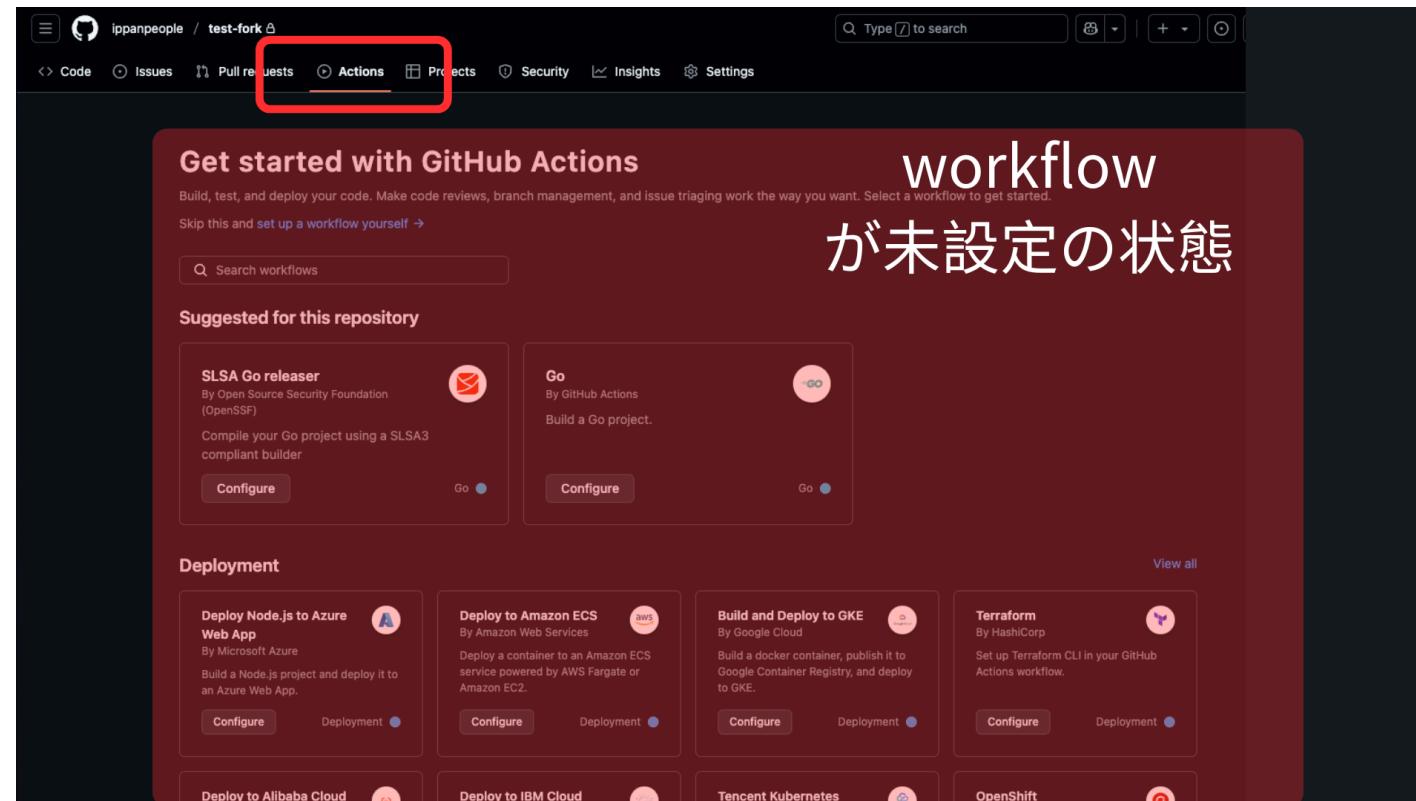
フォーク直後や新規リポジトリでは、GitHub Actions ワークフローが無効化されている場合があるので、必要に応じて有効化する必要があります。



# Workflow の有効化 SKIP

## 1. ワークフローの初期状態を確認

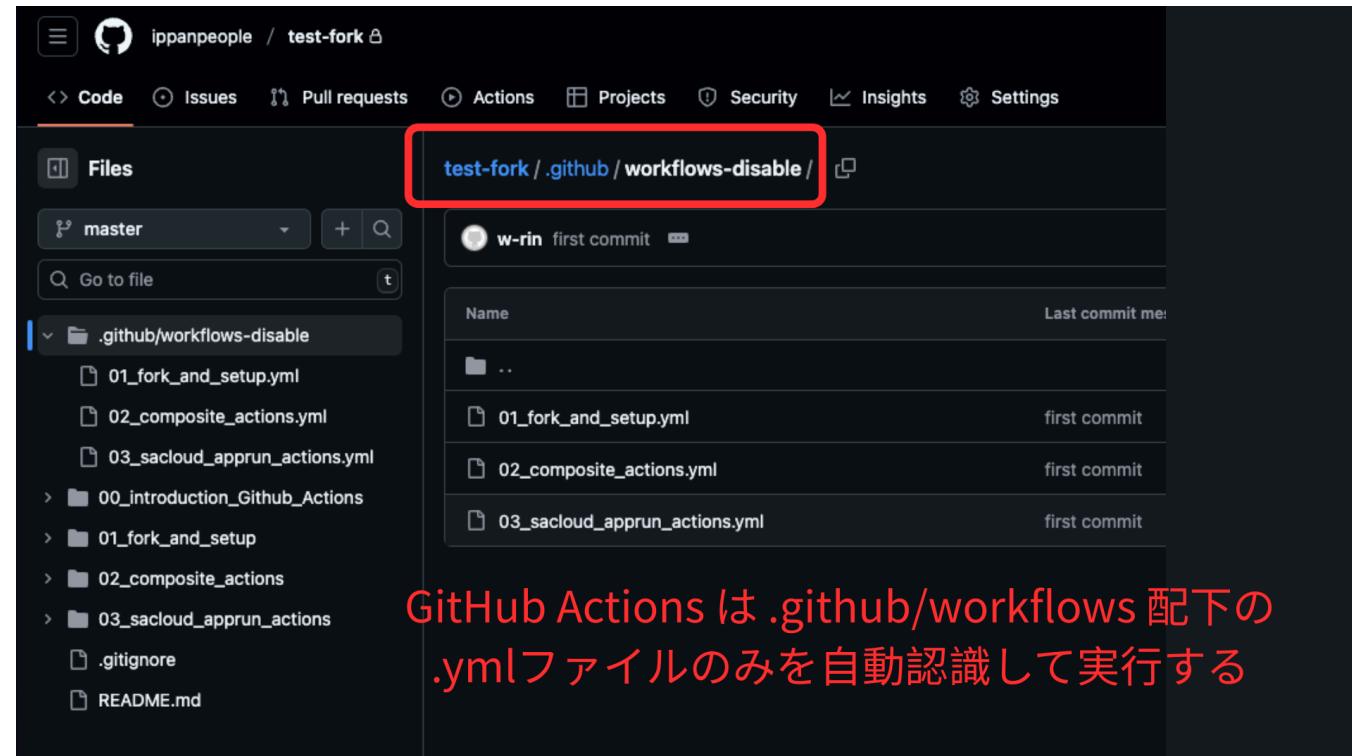
- 有効化されてない状態



# Workflow の有効化 SKIP

## 2. ワークフローを有効化する必須要件

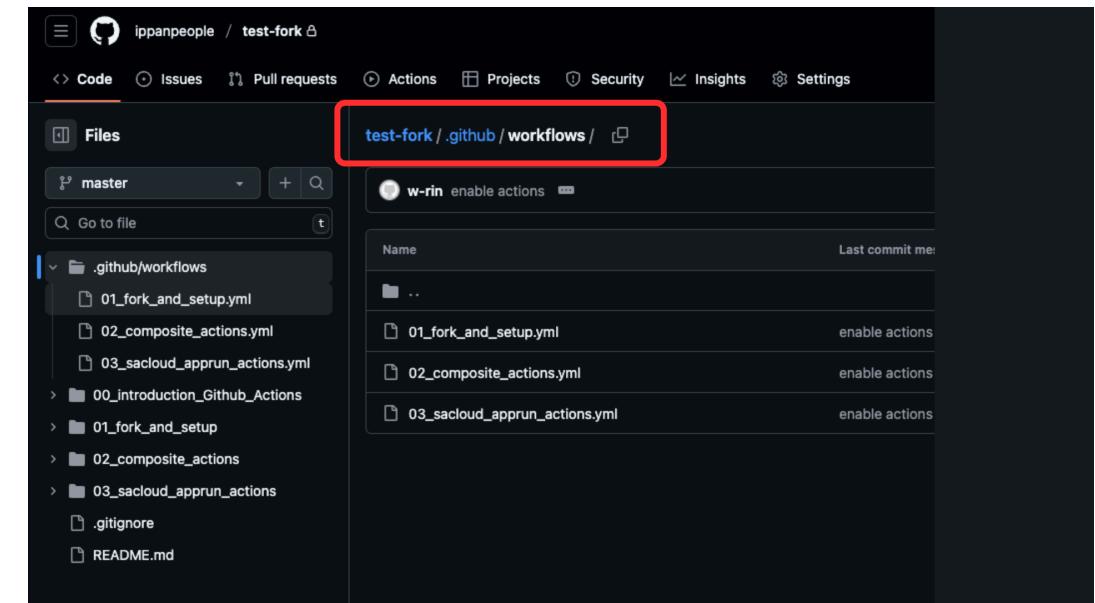
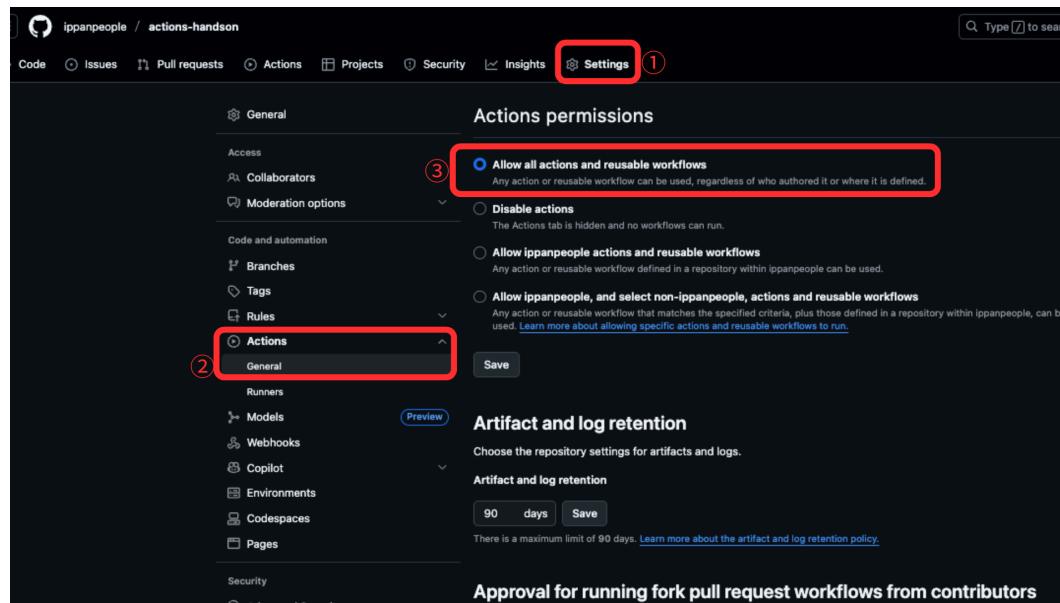
- リポジトリの設定で Actions のパーミッションが適切に設定
- `.github/workflows` ディレクトリ内にワークフロー定義ファイルが存在



# Workflow の有効化 SKIP

## 3. ワークフローを有効化しよう

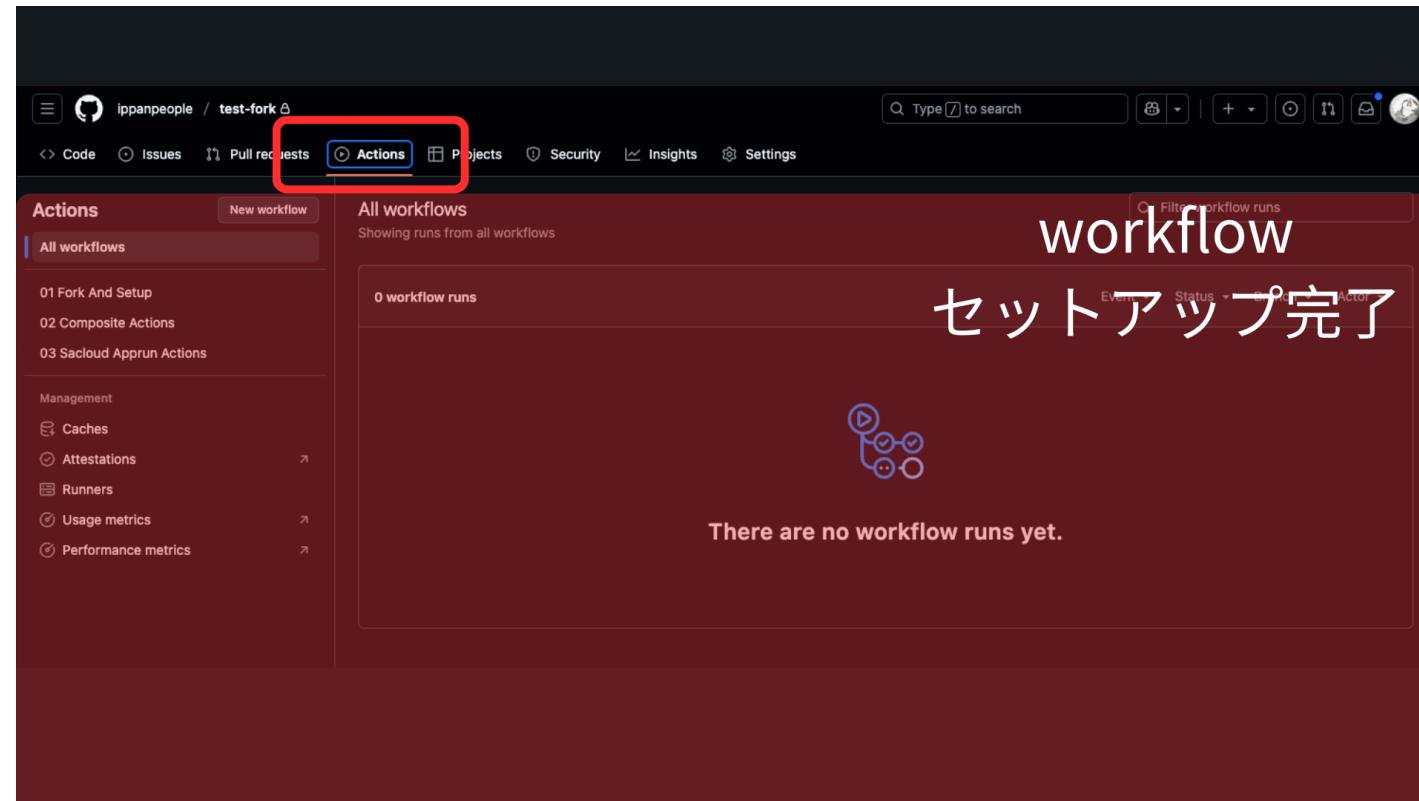
- Settings > Actions > General に移動し Actions permissions を Allow all actions and reusable workflows に設定
- 既存の .github/workflows-disable を正しくリネームして .github/workflows にする



# Workflow の有効化

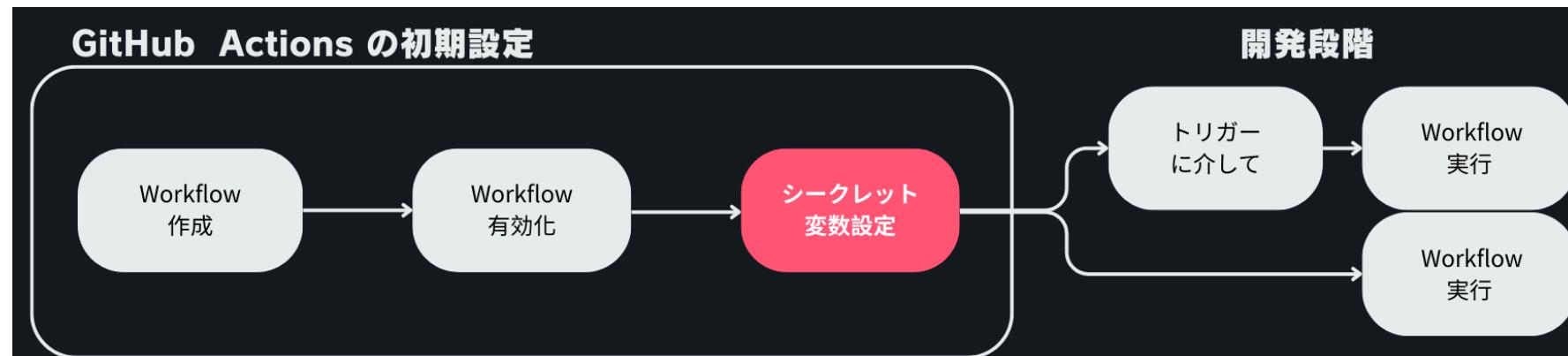
## 4. 有効化後の状態を確認

- 有効化が完了すると、ワークフローが実行可能な状態になる



# シークレットと変数の設定

GitHub Actions では、外部サービスへの認証情報や個人情報を安全に管理するために **シークレット (Secrets)** と **変数 (Variables)** を利用します。シークレットは主にパスワードや API キーなどの機密情報を、変数はワークフロー内で再利用したい値を格納します。これらはリポジトリの `Settings > Secrets and variables` から設定できます。これから今回のハンズオンに使用するシークレットと変数を設定していきます。

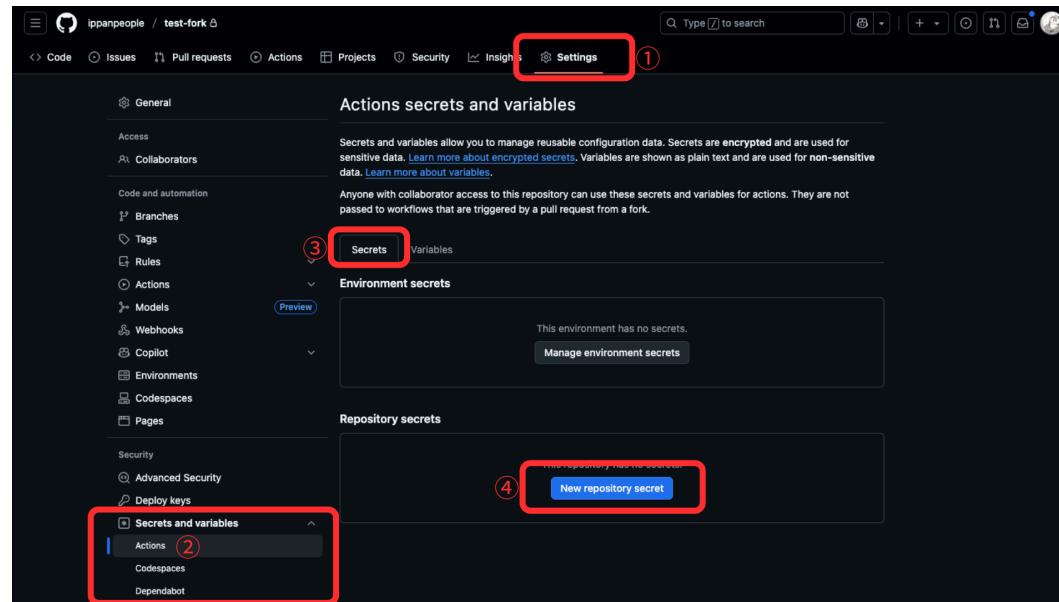


# シークレットと変数の設定

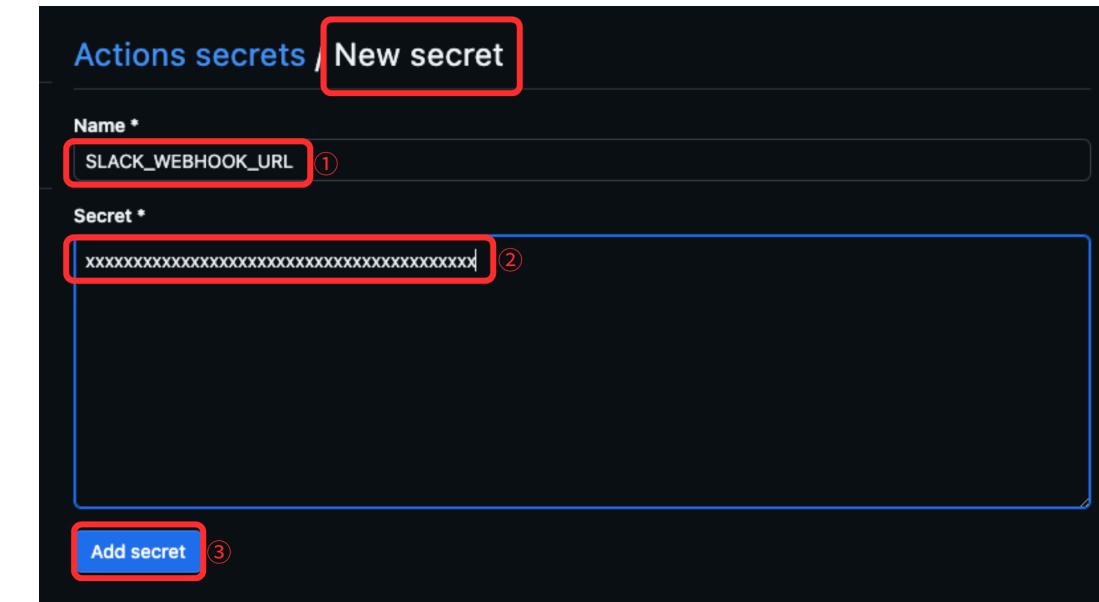
ハンズオン

## 1. シークレットの設定

Name	Value
SLACK_WEBHOOK_URL	Slack の Incoming Webhook URL



This screenshot shows the GitHub Actions secrets and variables settings page for a repository named 'ippnpeople/test-fork'. The left sidebar has a red box around the 'Secrets and variables' section, with a red number ② next to it. The main content area shows the 'Actions secrets and variables' section with a red box around the 'Secrets' tab, and a red number ① next to the 'Settings' button at the top. A red box also surrounds the 'New repository secret' button at the bottom right, with a red number ④ next to it. The URL in the browser bar is [https://github.com/ippnpeople/test-fork/settings/secrets/actions](#).

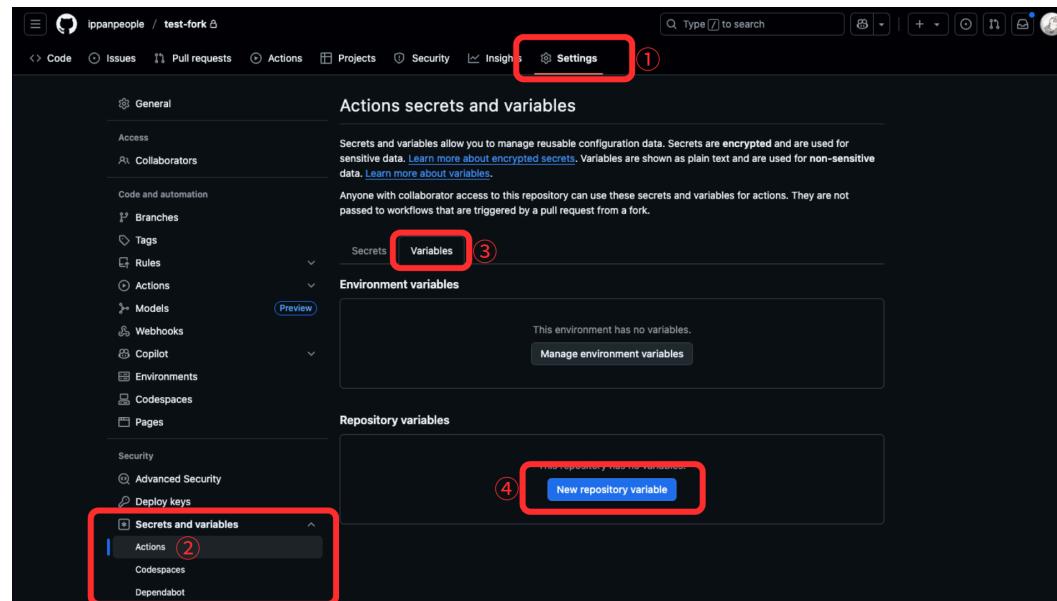


This screenshot shows the 'New secret' creation page for GitHub Actions secrets. The title is 'Actions secrets / New secret'. The 'Name \*' field contains 'SLACK\_WEBHOOK\_URL' with a red box and a red number ①. The 'Secret \*' field contains a long string of X's ('xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx') with a red box and a red number ②. At the bottom is a red box around the 'Add secret' button, with a red number ③ next to it. The URL in the browser bar is [https://github.com/ippnpeople/test-fork/actions/secrets/new](#).

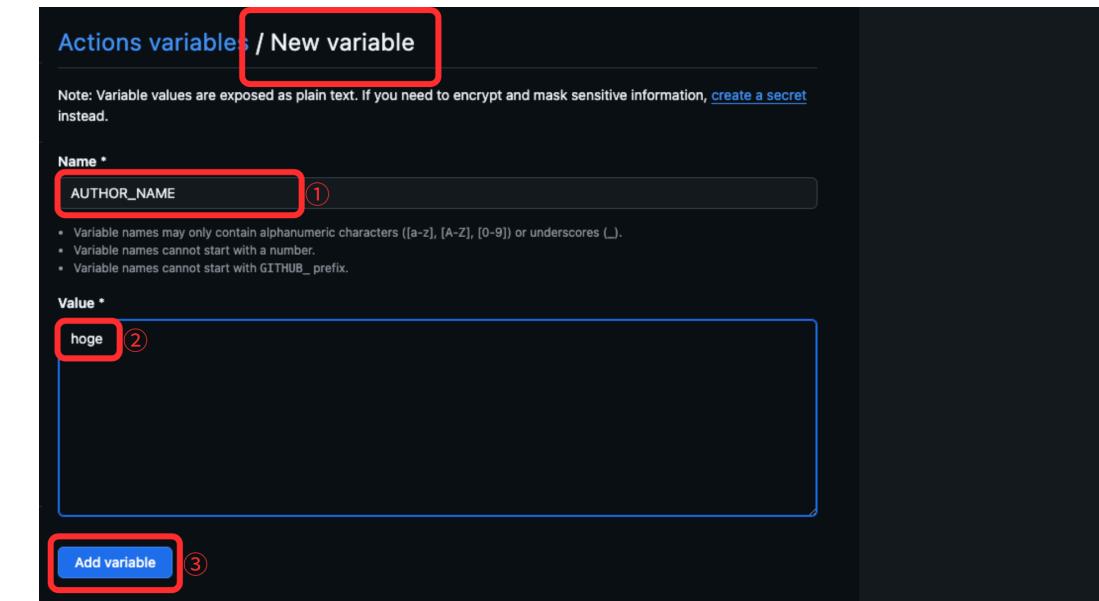
# シークレットと変数の設定 ハンズオン

## 2. 変数の設定

Name	Value
AUTHOR_NAME	自分の名前
REPOSITORY	自分の GitHub リポジトリリンク
MESSAGE	メッセージ内容(例: "Hello, World!")



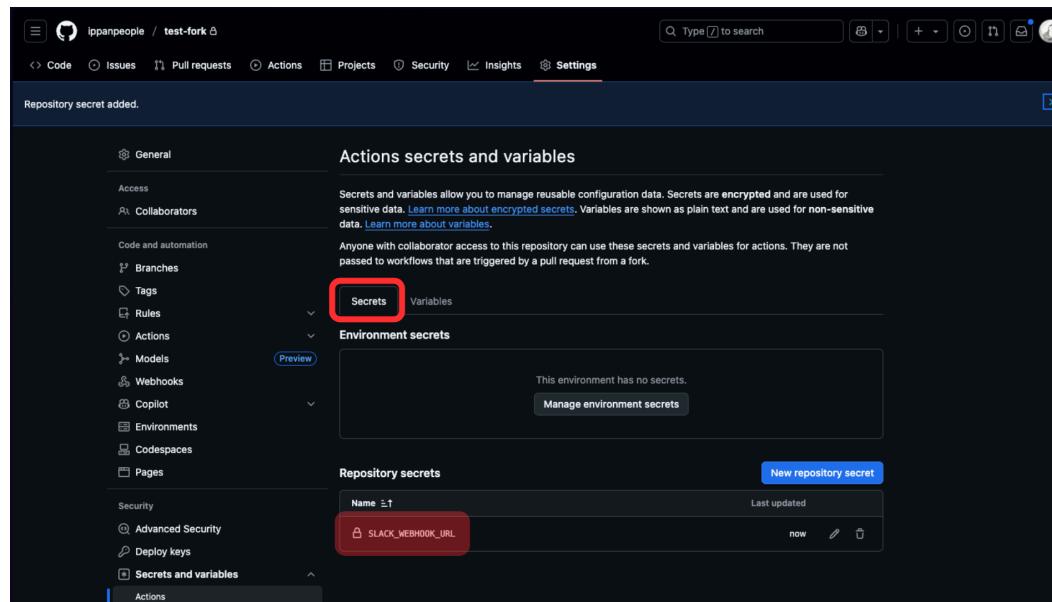
This screenshot shows the GitHub Actions secrets and variables settings page for a repository named 'ippnpeople/test-fork'. The left sidebar has a red box around 'Secrets and variables' under 'Actions'. The main area shows the 'Actions secrets and variables' section with a red box around the 'Variables' tab. A red box also surrounds the 'New repository variable' button at the bottom right. Numbered circles (1, 2, 3, 4) point to the 'Settings' button, the 'Variables' tab, the 'New repository variable' button, and the 'New repository variable' button respectively.



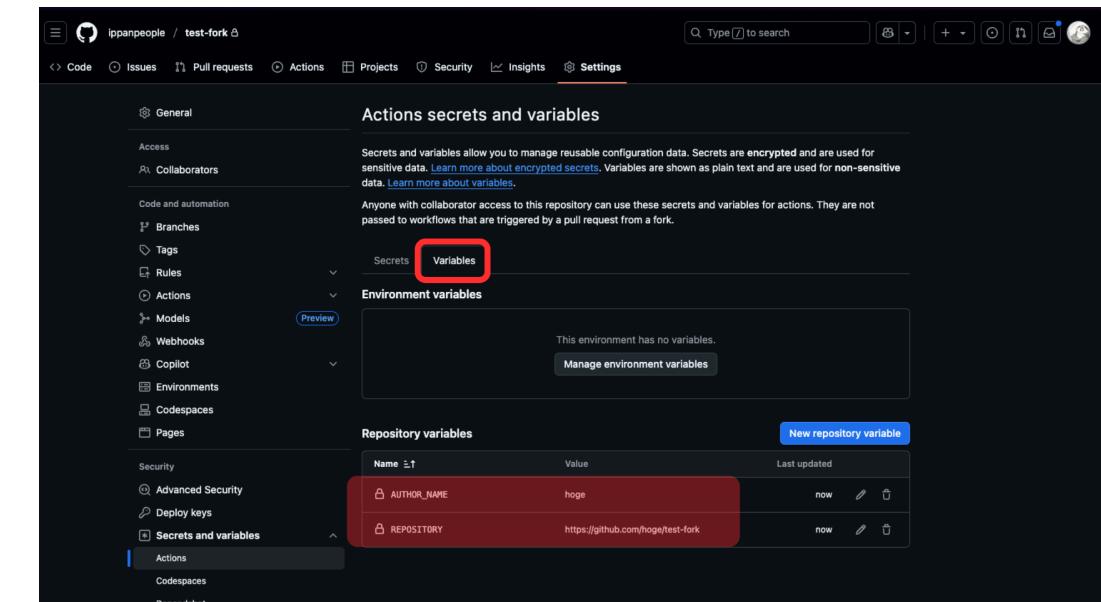
This screenshot shows the 'Actions variables / New variable' dialog. It includes a note about variable values being exposed as plain text. The 'Name' field contains 'AUTHOR\_NAME' with a red box and circle (1). The 'Value' field contains 'hoge' with a red box and circle (2). At the bottom is a blue 'Add variable' button with a red box and circle (3).

# シークレットと変数の設定

## 3. 設定後の確認: 設定が正しく反映されているか確認



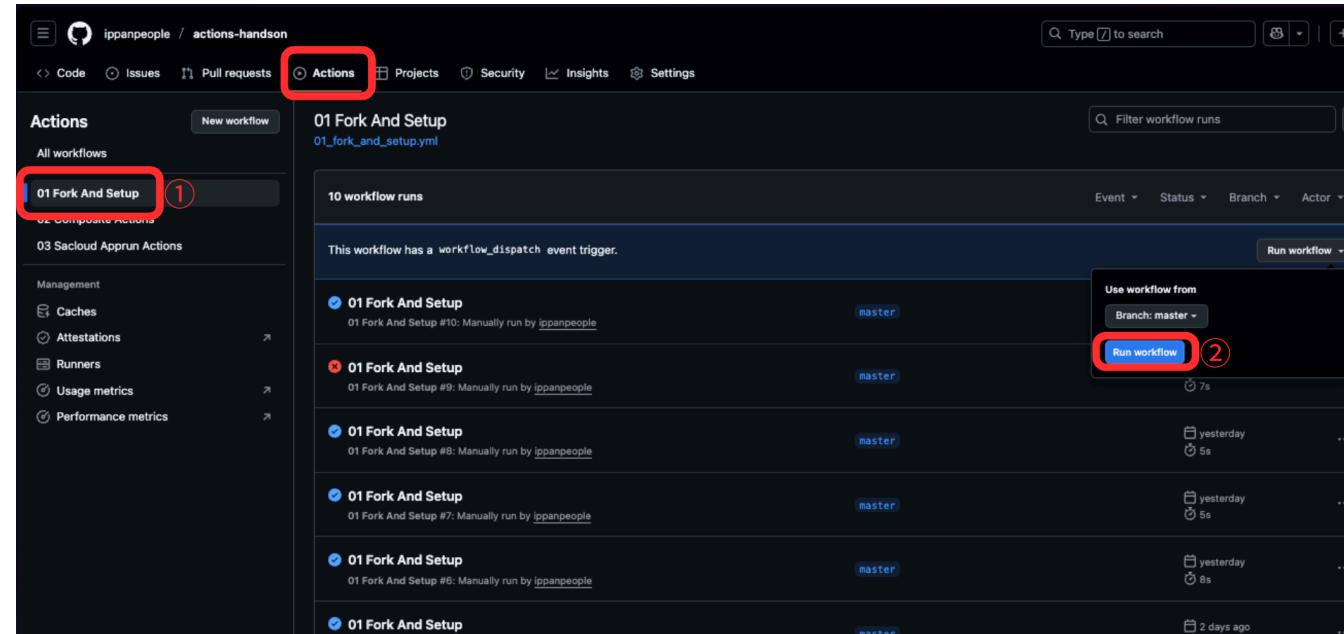
The screenshot shows the GitHub Actions settings page for the repository 'ippnpeople/test-fork'. The 'Actions secrets and variables' section is active. On the left, the 'Secrets' tab is highlighted with a red box. In the main area, under 'Repository secrets', there is a single entry named 'SLACK\_WEBHOOK\_URL' with the value 'https://github.com/hoge/test-fork'. A 'New repository secret' button is visible.



The screenshot shows the GitHub Actions settings page for the repository 'ippnpeople/test-fork'. The 'Actions secrets and variables' section is active. On the left, the 'Variables' tab is highlighted with a red box. In the main area, under 'Repository variables', there is one entry named 'REPOSITORY' with the value 'https://github.com/hoge/test-fork'. A 'New repository variable' button is visible.

# ワークフローの実行 ハンズオン

1. ワークフローの選択: リポジトリの **Actions** タブに移動し、**01 Fork And Setup** のワークフローを選択
2. ワークフローの実行: **Run workflow** ボタンをクリック
3. 実行の確認: ワークフローが正常に実行されると、Slack チャンネルにメッセージが送信される。これにより、GitHub Actions のセットアップが正しく行われたことを確認できる



# Composite Actions の運用 SKIP

02\_composite\_actions



# Composite Actions とは? SKIP

`01_fork_and_setup` では、`run` でコマンドを実行したが、より複雑な処理を定義したい場合、ワークフローはスパゲッティ化してしまうことがあります。

Composite Actions は、複数のシェルステップを組み合わせて作成されるカスタムアクションです。YAML で記述され、GitHub Actions のワークフロー内で再利用可能なロジックを提供します。これにより、複雑な処理を簡潔にまとめ、再利用性を高めることができます。

# 前のステップとの比較 SKIP

`uses` で公開されている Composite Actions を呼び出し、`with` でパラメータを指定することで、複雑な処理を簡潔にまとめることができる

```

Code Blame 35 lines (33 loc) · 1.34 KB

1 name: 01 Fork And Setup
2
3 on:
4   workflow_dispatch:
5
6 jobs:
7   notify:
8     runs-on: ubuntu-latest
9     steps:
10    - name: Notify to Slack
11      run: |
12        TS=$(date +%s)
13        cat <>EOF > payload.json
14        {
15          "channel": "dev-インターネット_ハンズオン",
16          "thread_ts": "1753676783.810409",
17          "attachments": [
18            {
19              "fallback": "ハンズオン 01_fork_and_setup の通知",
20              "color": "#215EBF",
21              "pretext": "STEP1: fork and setup ✅ <!channel>",
22              "author_name": ":github_white: ${vars.AUTHOR_NAME}",
23              "title": "github Repository",
24              "title_link": "${vars.REPOSITORY}",
25              "text": ":speech_balloon: (適宜変更してください)",
26              "footer": "インターネット_ハンズオン BOT",
27              "footer_icon": "https://qiita-user-contents.imgix.net/https%3A%2F%2Fqiita-image-store.s3.ap-northeast-1.
28              "ts": $TS
29            }
30          ]
31        }
32      EOF
33      curl -X POST -H 'Content-type: application/json' \
34      --data @payload.json \
35      ${secrets.SLACK_WEBHOOK_URL}

```

Before

```

Code Blame 18 lines (16 loc) · 527 Bytes

1 name: 02 Composite Actions
2
3 on:
4   workflow_dispatch:
5
6 jobs:
7   notify:
8     runs-on: ubuntu-latest
9     steps:
10    - name: Notify to Slack
11      uses: ippanpeople/slack-notify-action@v0.0.4
12      with:
13        webhook_url: ${secrets.SLACK_WEBHOOK_URL}
14        thread_ts: "1753676783.810409"
15        pretext: "STEP2: composite actions ✅ <!channel>"
16        author_name: ":github_white: ${vars.AUTHOR_NAME}"
17        repository: "${vars.REPOSITORY}"
18        message: ":speech_balloon: (適宜変更してください)"

```

After

# 実行してみよう

SKIP

`02_composite_actions.yml` は `01_fork_and_setup.yml` の内容を Composite Actions を使って書き直したものだけなので、各自実行して、動作を確認してください。

1. ワークフローの選択: リポジトリの `Actions` タブに移動し、`Composite Actions` の `ワークフロー` を選択
2. ワークフローの実行: `Run workflow` ボタンをクリック
3. 実行の確認: ワークフローが正常に実行されると、Slack チャンネルにメッセージが送信される。これにより、GitHub Actions のセットアップが正しく行われたことを確認できる

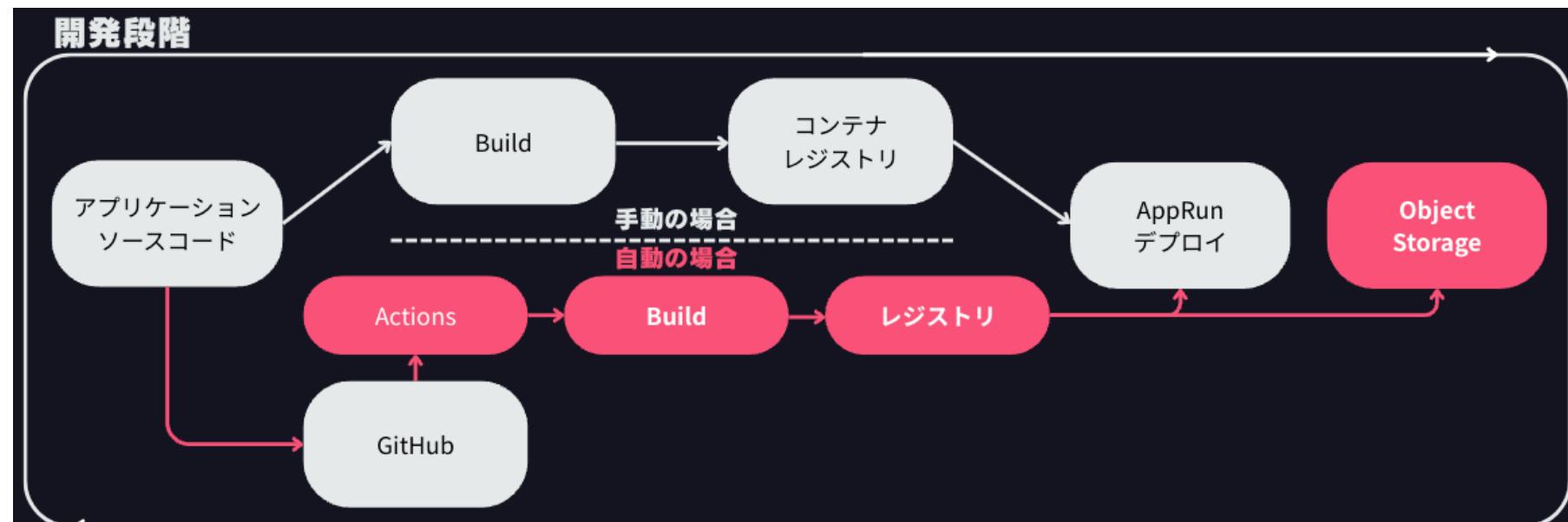
# sacloud\_apprun\_actions を使った AppRun へのデプロイ

## 03\_sacloud\_apprun\_actions



# sacloud\_apprun\_actions とは？

`sacloud_apprun_actions` は、Go アプリケーションをさくらの AppRun にデプロイするときのフローを自動化するための Composite Actions です。デプロイする以外、データ永続化のため、登録されたオブジェクトストレージのバケットに SQLite レプリカの作成機能も含まれており、アプリケーション再デプロイ後もデータの永続化ができます。



# sacloud\_apprun\_actions の使い方

1. ソースコードの準備: `sacloud_apprun_actions` は Go アプリケーションに対して、ソースコードをビルト用の Dockerfile を提供しているので、リポジトリに Dockerfile を配置しなくても、ソースコードを用意することで利用可能

# sacloud\_apprun\_actions の使い方

2. ワークフロー例: `sacloud_apprun_actions` を使用する時下記のようなタスクを作成し、パラメーターを設定することで利用可能

# sacloud\_apprun\_actions の使い方

```
- name: Goアプリをデプロイ
  uses: ippnpeople/sacloud_apprun-action@v0.0.4
  with:
    use-repository-dockerfile: false
    app-dir: ./03_sacloud_apprun_actions
    sakura-api-key: ${{ secrets.SAKURA_API_KEY }}
    sakura-api-secret: ${{ secrets.SAKURA_API_SECRET }}
    container-registry: ${{ secrets.REGISTERY }}
    container-registry-user: ${{ secrets.REGISTERY_USER }}
    container-registry-password: ${{ secrets.REGISTERY_PASSWORD }}
    port: '8080'
    # SQLite + Litestream を使う場合は以下も指定
    object-storage-bucket: ${{ secrets.STORAGE_BUCKET_NAME }}
    object-storage-access-key: ${{ secrets.STORAGE_ACCESS_KEY }}
    object-storage-secret-key: ${{ secrets.STORAGE_SECRET_KEY }}
    sqlite-db-path: ./data/app.db
    litestream-replicate-interval: 10s
```

# sacloud\_apprun\_actions の使い方 ハンズオン

## 3. Secrets の設定: リポジトリの設定で必要な GitHub Actions シークレットを登録しよう

Name	Value
REGISTRY	コンテナレジストリの URL
REGISTRY_USER	コンテナレジストリのユーザー名
REGISTRY_PASSWORD	コンテナレジストリのパスワード
SAKURA_API_KEY	さくらの API キー
SAKURA_API_SECRET	さくらの API シークレット
STORAGE_BUCKET_NAME	オブジェクトストレージのバケット名
STORAGE_ACCESS_KEY	オブジェクトストレージのアクセスキー
STORAGE_SECRET_KEY	オブジェクトストレージのシークレットキー

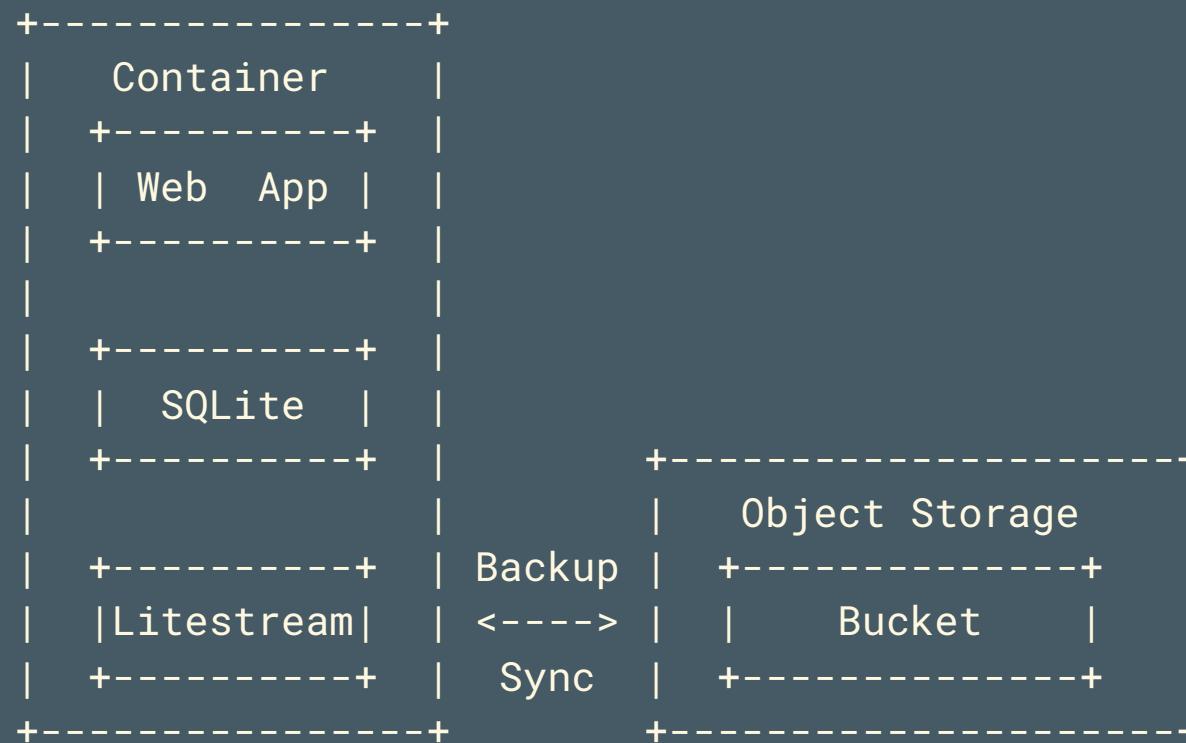
## ハンズオン

# sacloud\_apprun\_actions の使い方

3. ワークフローの実行: ワークフローを手動で実行してみよう
  - a. ワークフローの選択: リポジトリの **Actions** タブに移動し、  
03 Sacloud Apprun Actions の **ワークフロー** を選択
  - b. ワークフローの実行: **Run workflow** ボタンをクリック
  - c. 実行の確認: ワークフローが正常に実行されると、Slack チャンネルにメッセージが送信される。これにより、GitHub Actions のセットアップが正しく行われたことを確認できる

# データ永続化の実践方法

AppRun は **ステートレス** なため、デプロイのたびにアプリケーションが再起動されます。  
sacloud\_apprun\_actions はデータ永続化の課題を解決するため、  
**SQLite** と **Litestream** を利用し、アプリ再起動後もデータが保持されるようにします。



# データ永続化の実践方法

## ⚠ 注意:

SQLite + Litestream を利用する際は、システム設計の妥当性に注意してください。SQLite は小規模用途に適していますが、高負荷や大量データではパフォーマンス上の制約があります。特に、TPS (Transactions Per Second) や、QPS (Queries Per Second) などの観点で制約が発生しやすいです。なので、データの安全性・一貫性 (Consistency) に注意が必要です。システム要件に応じて、適切なデータベースやストレージ方式の選定を検討してください。

# 実際の開発フローから Actions のメリットを体験

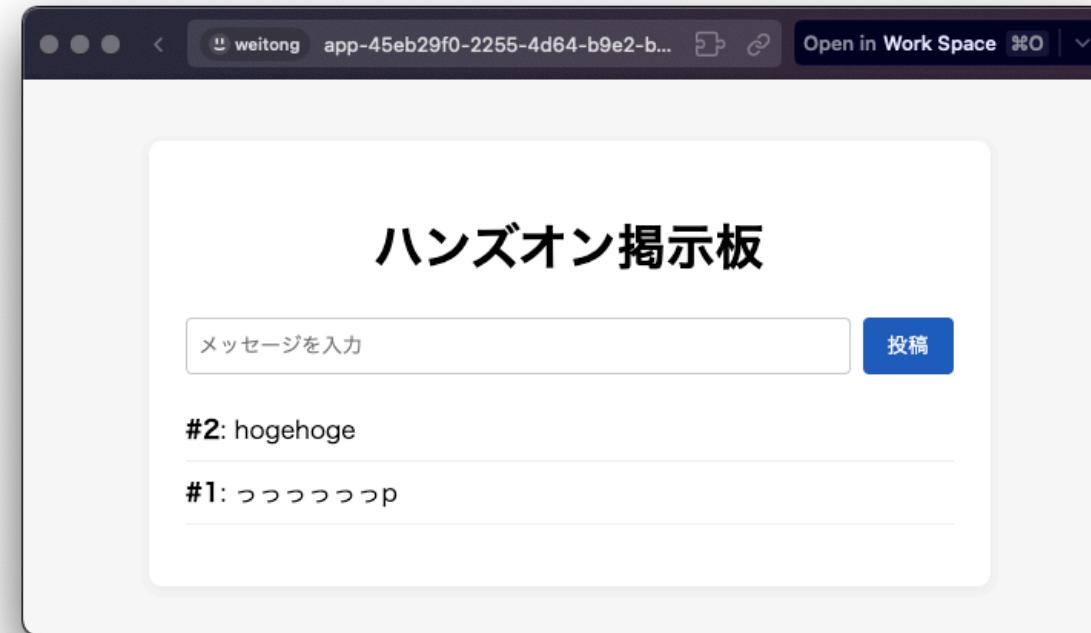
改めて強調するが、AppRun のステートレスの特性で、0スケールや、バージョンアップに応じて再デプロイする際に、データが失われてしまうことがあります。もちろん外部のデータベースを使っても解決できる問題ですが、気軽に開発していく人に対して、かなり不便なところです。

なので、これから実際の開発フローに介して、`sacloud_apprun_actions` を使ったときのメリットを体験してみましょう。

補足：なぜオブジェクトストレージを使うのか？基本的にマネジドなDBより、安価で AppRun のユースケースに親和性が高いためです。

# 実際の開発フローから Actions のメリットを体験

- データの登録: `sacloud_apprun_actions` でデプロイされた各自のアプリにアクセスし、メッセージを投稿



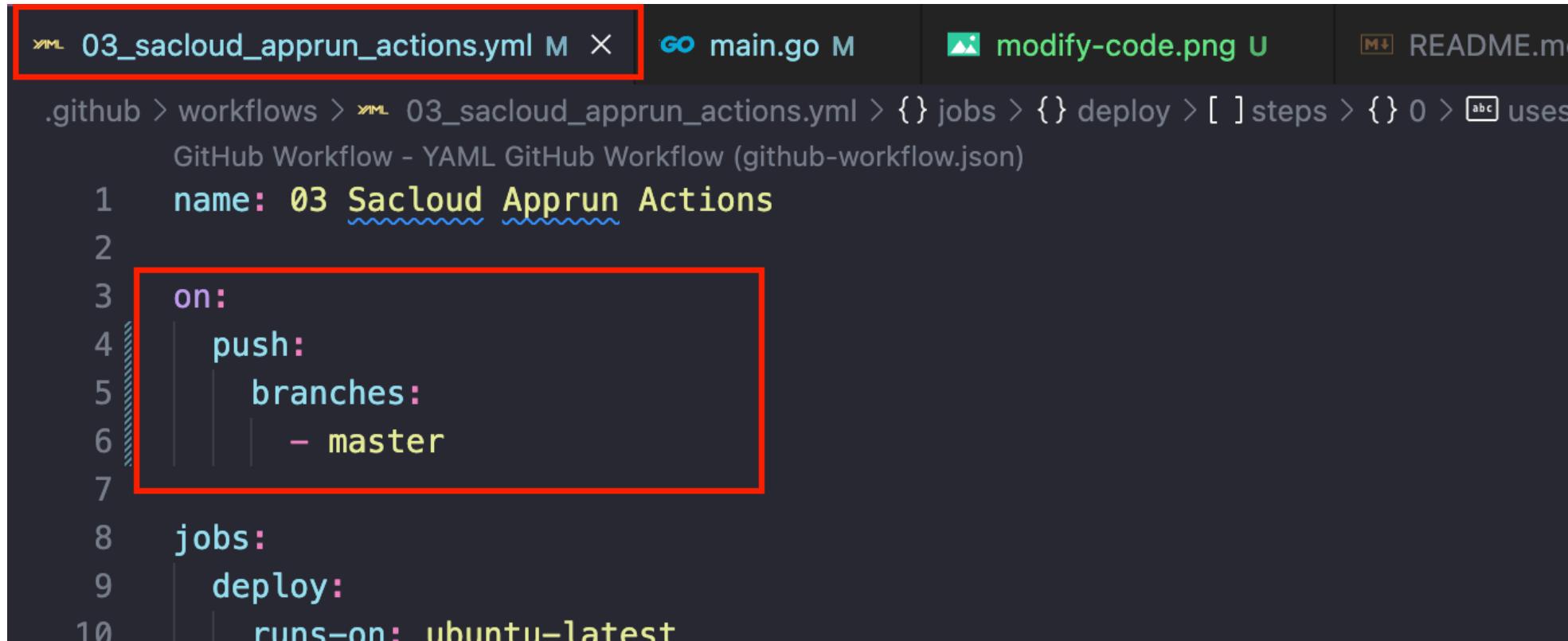
# 実際の開発フローから Actions のメリットを体験

## 2. ソースコードの修正: `03_sacloud_apprun_actions` のソースコードを修正

```
1, M      31     http.HandleFunc("/", func(w http.ResponseWriter, r *http.Request) {
2   es      32       w.Header().Set("Content-Type", "text/html; charset=utf-8")
3   ml .github... 33       fmt.Fprintln(w, `<!DOCTYPE html><html lang='ja'><head><meta charset='utf-8'><title>掲示板</title>
4   posite_acti... 34       body { font-family: sans-serif; background: #f7f7f7; }
5   oud_appru... 35       .container { max-width: 500px; margin: 40px auto; background: #fff; border-radius: 8px; box-shad
6   mposite_ac... 36       h1 { text-align: center; }
7   g         37       ul { padding: 0; }
8   s         38       li { list-style: none; border-bottom: 1px solid #eee; padding: 8px 0; }
9   ctions ● 39       form { display: flex; gap: 8px; margin-bottom: 16px; }
10  M        40       input[type=text] { flex: 1; padding: 8px; border: 1px solid #ccc; border-radius: 4px; }
11  terraform 41       button { padding: 8px 16px; border: none; background: #215EBF; color: #fff; border-radius: 4px; }
12  ●        42       button:hover { background: #174a8c; }
13  u        43       </style></head><body><div class='container'>`)
14  ●        44       fmt.Fprintln(w, `<h1>ハングオン掲示板 v2.0.0</h1>`)
15  ●        45       fmt.Fprintln(w, `<form method='POST' action='/add'><input type='text' name='msg' placeholder='
16  ●        46       fmt.Fprintln(w, `<ul>`)
17  ●        47       rows, err := db.Query("SELECT id, content FROM messages ORDER BY id DESC")
18  ●        48       if err != nil {
19  ●        49           fmt.Fprintf(w, "<li style='color:red;'>%s</li>", err.Error())
20  ●        50       } else {
21  ●        51           defer rows.Close()
22  ●        52           for rows.Next() {
23  ●        53               var id int
24  ●        54               var content string
25  ●        55               rows.Scan(&id, &content)
26  ●        56               fmt.Fprintf(w, "<li><b>%d</b>: %s</li>", id, content)
```

# 実際の開発フローから Actions のメリットを体験

3. トリガーの変更: `03_sacloud_apprun_actions.yml` のトリガーを `push` に変更



```
.github > workflows > 03_sacloud_apprun_actions.yml > {} jobs > {} deploy > [ ] steps > {} 0 > abc uses
GitHub Workflow - YAML GitHub Workflow (github-workflow.json)
1   name: 03 Sacloud Apprun Actions
2
3   on:
4     push:
5       branches:
6         - master
7
8   jobs:
9     deploy:
10      runs-on: ubuntu-latest
```

# 実際の開発フローから Actions のメリットを体験

4. アプリケーションのバージョンアップ: 修正をプッシュしたら、GitHub Actions が自動的に実行し、新バージョンを AppRun に再デプロイ

# 実際の開発フローから Actions のメリットを体験

## 5. 動作確認 1

- AppRunのバージョン情報確認: AppRun のバージョン情報を確認し、最新のバージョンがデプロイされていることを確認

 AppRun β

アプリケーション (1)			
<input type="checkbox"/> アプリケーション名	ステータス	作成日時	操作
<a href="#">actions-handson</a>	<span style="color: green;">● 正常</span>	2025-07-30 14:34:34	...

 AppRun β

actions-handson			
コンテナレジストリ	アプリケーションID	作成日時	ポート
<small>β版における注意事項と制限事項</small>	45eb29f0-2255-4d64-b9e2-b1cf6bc3cc34	2025-07-30 14:34:34	8080
	最小スケール数	最大スケール数	リクエストタイムアウト
	0	1	300 seconds
	ステータス	公開URL	リソースID
	<span style="color: green;">● 正常</span>	<a href="https://app-45eb29f0-2255-4d64-b9e2-b1cf6bc3cc34.ingress.apprun.sakura.ne.jp">https://app-45eb29f0-2255-4d64-b9e2-b1cf6bc3cc34.ingress.apprun.sakura.ne.jp</a>	113701971545

フィードバック

バージョン (3) ?

バージョン名	ステータス	トラフィック	作成日時
<a href="#">actions-handson-00003</a>	<span style="color: green;">● 正常</span>	100% (100% を最新にルーティング)	56分前
<a href="#">actions-handson-00002</a>	<span style="color: green;">● 正常</span>	0%	3時間前
<a href="#">actions-handson-00001</a>	<span style="color: green;">● 正常</span>	0%	2025-07-30 14:34:34

# 実際の開発フローから Actions のメリットを体験

## 5. 動作確認 2

- データの確認: アプリケーションに再度アクセスし、先ほど登録したデータが残っていることを確認



# 以上で GitHub Actions 説明は終了です

Actions を使った開発も体験してもらったので、ぜひイン  
ターンで活用

質問等ある際にお気軽に聞いてください

