

Kata

Continuous Integration with Github Actions

What is Github Actions?

What is Github Actions?

• Introduction

Github Actions provides a declarative way to integrate automation tasks per repository.

Configuration files are just committed into the respective repository and as such can be easily edited by every collaborator.

Common use-cases include:

- Testing
- Building and publishing artifacts
- Generating documentation
- Sending notifications on new versions, e.g. as Slack messages

What is Github Actions?

• Introduction

The Actions UI is integrated into Github and reachable from any Repository by clicking on the "Actions"-tab:

Workflows [New workflow](#)

All workflows

[Publish on Release](#)

[Test on Push](#)

Filter workflows

Event	Status	Branch	Actor
✓ Release 3.13.1	Publish on Release #5: Release 3.13.1 published by oktaiaut		20 hours ago 12m 18s
✓ Use 'sudo' to Switch Xcode Versions	Test on Push #11: Commit 2bc9307 pushed by oktaiaut	feature/build_using_swift...	20 hours ago 3m 44s
ⓘ Publish on Release	Publish on Release #4: published by oktaiaut		20 hours ago 2m 48s
✓ Use Correct Xcode Version	Test on Push #10: Commit a7ee2fd pushed by oktaiaut	feature/build_using_swift...	20 hours ago 3m 8s
✓ Publish on Release	Publish on Release #3: published by oktaiaut		20 hours ago 13m 7s
✓ Update Pod Version	Test on Push #9: Commit 592d173 pushed by oktaiaut	feature/build_using_swift...	20 hours ago 4m 59s
✓ Add Build Job using Swift 5.1	Test on Push #8: Commit 8bdb6ec pushed by oktaiaut	feature/build_using_swift...	20 hours ago 4m 55s

What is Github Actions?

- Introduction

Github Actions runs fully in the cloud, but provides you with escape hatches to host your on job runners.

This can be useful if services are deployed in a private network or jobs involve running tests on hardware devices.

Pricing might be an issue, too. While runners are available for the major OSs, Mac OS build minutes are four times more expensive than those on Linux.

How does it work?

How does it work?

- Workflows

Entrypoint into each CI process are `yml`-files called workflows.

Github automatically picks up on any workflow file under `.github/workflows/`:



Each file consist of one ore more triggers, which define the behavior of when certain jobs should be run.

These jobs in return are composed of several steps.

How does it work?

- Workflows

Simple example of a workflow file:

```
# workflow name
name: Test on Push

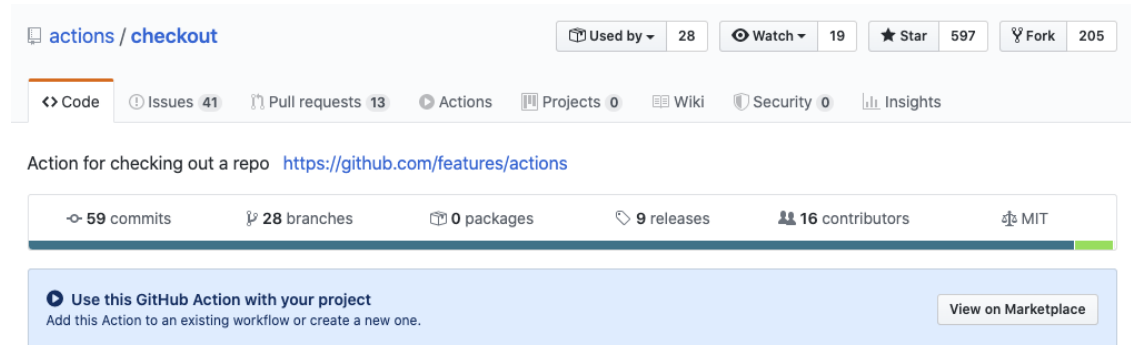
# workflow is executed on every push to a feature branch
on:
  push:
    branches:
      - feature/*

# jobs are run independently of one another
jobs:
  test:
    # target machine on which job is run
    runs-on: ubuntu-latest
    steps:
      # step runs a premade Action to checkout the repo
      - uses: actions/checkout@v2
      # custom steps that run in a shell
      - name: Install Dependencies
        run: |
          make dependencies.install
      - name: Run Tests
        run: |
          make test
```


How does it work?

- Workflows
- Actions

Single steps can be developed in isolation and packaged as so-called "Actions".



How does it work?

- Workflows
- Actions

Actions are reusable components for workflows, which can be semantically versioned.

This prevents breaking workflows should a major update occur on one of your actions.

```
steps:  
  # Actions are references by their repository name  
  # and version tag  
  - uses: actions/checkout@v2  
  # You can optionally name a step that uses an Action,  
  # otherwise the Action name is displayed  
  - name: Checkout Repository
```

How does it work?

- Workflows
- Actions
- Control Flow

Steps are executed in sequential order.

If one step fails, the following steps are skipped and the job is marked as failed.

The execution of steps and their logs can be viewed on the repository's page:

The screenshot shows a GitHub Actions workflow run for the job 'Build' on the 'feature/pipeline' branch. The workflow is titled 'Fix Workflow File' and was triggered by a push from user '2d0d77a'. The job 'build' is shown as successful. The right-hand pane displays the execution log for the 'Build / build' job, which succeeded 2 days ago in 2m 15s. The log shows a 'Set up job' step with sub-steps for runner version, operating system, virtual environment, and downloading actions. It also lists 'Run actions/checkout@v2', 'Test', 'Build', and 'Publish' as subsequent steps.

Fix Workflow File
feature/pipeline 2d0d77a

Build
on: push

✓ build

Build / build
succeeded 2 days ago in 2m 15s

Set up job

- 1 Current runner version: '2.169.1'
- 2 ▶ Operating System
- 6 ▶ Virtual Environment
- 10 Prepare workflow directory
- 11 Prepare all required actions
- 12 Download action repository 'actions/checkout@v2'
- 13 Download action repository 'actions/upload-artifact@v2'

▶ ✓ Run actions/checkout@v2

▶ ✓ Test

▶ ✓ Build

⌚ Publish

How does it work?

- Workflows
- Actions
- Control Flow

It is possible to run steps only on positive evaluation of a condition:

```
# Runs the steps only on the 'develop' branch
- name: Run Tests
  if: github.ref == 'refs/heads/develop'
  run: |
    make test
```

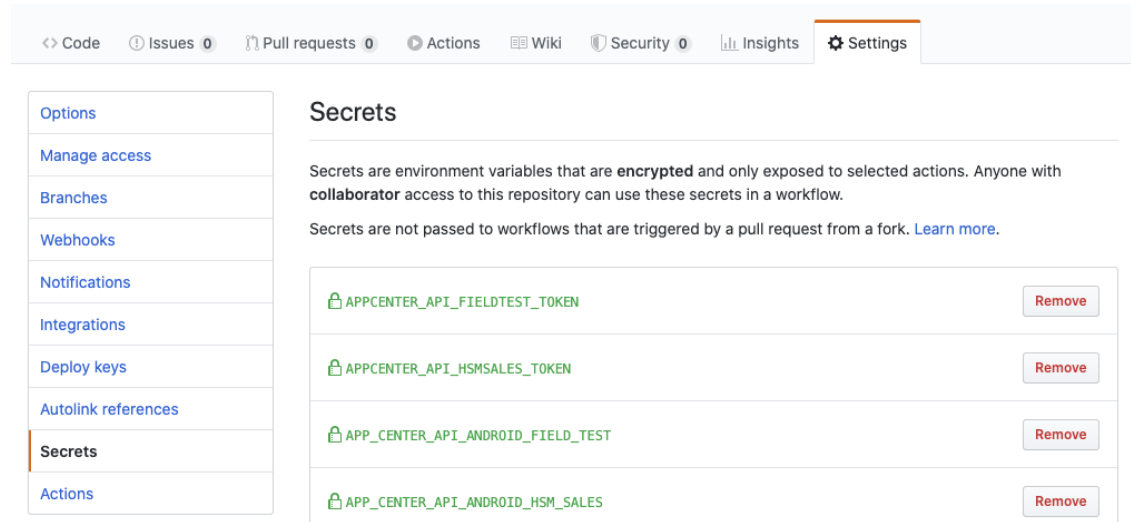
You can also run a step always, regardless if previous steps have failed (the job will still be shown as unsuccessful):

```
# Runs the steps even if previous steps failed
- name: Send Report to Slack
  if: ${ always() }
  run: |
    make slack.send_report
```

How does it work?

- Workflows
- Actions
- Control Flow
- Secrets

Secrets can be safely stored per repository and referenced in workflow steps.



The screenshot shows the GitHub repository settings interface. At the top, there's a navigation bar with tabs for Code, Issues, Pull requests, Actions, Wiki, Security, Insights, and Settings. The Settings tab is active. On the left, a sidebar menu lists various settings categories: Options, Manage access, Branches, Webhooks, Notifications, Integrations, Deploy keys, Autolink references, Secrets (highlighted), and Actions. The main content area is titled 'Secrets' and contains explanatory text: 'Secrets are environment variables that are **encrypted** and only exposed to selected actions. Anyone with **collaborator** access to this repository can use these secrets in a workflow.' and 'Secrets are not passed to workflows that are triggered by a pull request from a fork. [Learn more.](#)'. Below this, a table lists four secrets, each with a lock icon, the secret name, and a 'Remove' button.

Secret Name	Action
APPCENTER_API_FIELDTEST_TOKEN	Remove
APPCENTER_API_HSMSALES_TOKEN	Remove
APP_CENTER_API_ANDROID_FIELD_TEST	Remove
APP_CENTER_API_ANDROID_HSM_SALES	Remove

There are additional secrets that are automatically injected for each job, the most frequently used being `GITHUB_TOKEN`.

`GITHUB_TOKEN` authenticates the actor running the jobs against Github APIs, e.g. for publishing packages.

How does it work?

- Workflows
- Actions
- Control Flow
- Secrets

Example of using the `GITHUB_TOKEN` secret to login to Docker private docker registry:

```
jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - name: Login to Docker Registry
        uses: azure/docker-login@v1
        with:
          login-server: docker.pkg.github.com
          username: $GITHUB_ACTOR
          # Access the Github actor's access token
          password: ${ secrets.GITHUB_TOKEN }
```

Kata

Kata

There aren't any requirements aside from having Git installed.

• Prerequisites

If you want to run the project's tasks locally, make sure you have the following installed as well:

- make
- Docker

To get started fork the following repository:

<https://github.com/oktinaut/kata-github-actions>

Kata

- Prerequisites

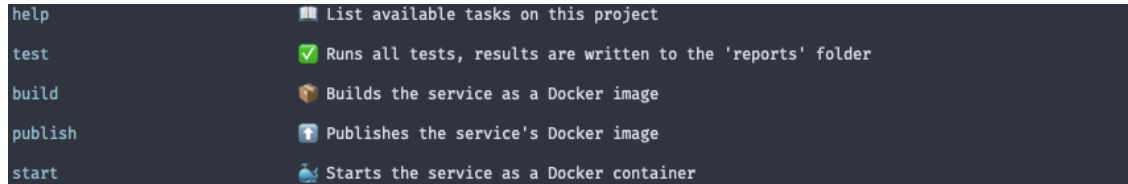
This project includes a simple Node.js server application.

It is our goal to automate tasks such as testing and building.

- Project

To help you getting started, the necessary tasks are already included in the project.

Run `make` inside the projects root folder to get a listing of all available tasks:



```
help          📖 List available tasks on this project
test          ✔ Runs all tests, results are written to the 'reports' folder
build         📦 Builds the service as a Docker image
publish       📤 Publishes the service's Docker image
start         🚢 Starts the service as a Docker container
```

Use `make <task-name>` to execute one.

Kata

- Prerequisites
- Project
- Tasks

Exercise 1

- Create a job that tests and build the application
- This job should run if changes are pushed to both the `develop` and `feature/` branches
- **BONUS:** Add a badge to display the build status in the projects `README.md`
- **BONUS:** Find a way to upload the test results as artifacts to Github

Exercise 2

- Create a job that publishes the server as a Docker image to the Github Packages registry
- This job should run on just the `develop` branch
- **BONUS:** Using conditional steps, integrate it all into the job from **exercise 1**

Kata

- Prerequisites

Github Actions Reference:

<https://help.github.com/en/actions/reference/workflow-syntax-for-github-actions>

- Project

<https://help.github.com/en/actions/reference/events-that-trigger-workflows>

- Tasks

Collection of Actions by Github:

<https://github.com/actions>

- Resources

Marketplace for custom Actions:

<https://github.com/marketplace>

Happy Hacking!