

.NET 6 & GitHub

Matteo Tumiati

Senior DevOps Engineer @icubedsrl

Microsoft MVP

matteot@aspitalia.com | @xtumiox

.NET Conference
Italia 2021



.NET

Agenda

Introduction to GitHub Actions

Apply the business needs

How .NET comes into play

GitHub Apps

GitHub Actions

GitHub Actions are a way used to automate tasks within software development life cycle in GitHub

GitHub Actions are event-driven, meaning that workflows are executed once a specified event has occurred

For example, every time someone creates a pull request for a repository, you can automatically run a command that executes a software testing script

Very similar to what a pipeline is in Azure DevOps or in Jenkins, Octopus and so on...

Dictionary

Actions are standalone commands that are combined into *steps* to create a *job*. Actions are the smallest portable building block of a workflow

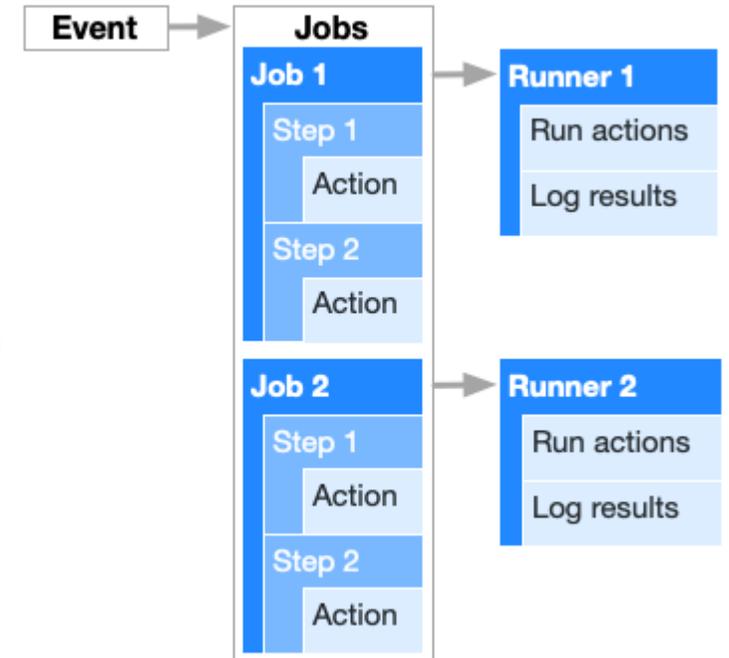
A **step** is an individual task that can run commands in a *job*

A **job** is a set of *steps* that execute on the same runner

An **event** is a specific activity that triggers a *workflow*

The **workflow** is an automated procedure (written in YAML) that you add to your repository within `.github/workflows`

A **runner** is a server that has the GitHub Actions runner application installed and that can execute the workflow



Creating the first workflow

```
name: .NET
```

```
on:  
  push:  
    branches: [ main ]  
  pull_request:  
    branches: [ main ]
```

```
jobs:
```

```
  build:  
    runs-on: ubuntu-latest  
    steps:  
      - uses: actions/checkout@v2  
      - name: Setup .NET  
        uses: actions/setup-dotnet@v1  
        with:  
          dotnet-version: 6.0.x  
      - name: Restore dependencies  
        run: dotnet restore  
      - name: Build  
        run: dotnet build --no-restore
```

Event

Job

Step

Action

Runner

What if I cannot find what I need?

Suppose that:

- I need to deploy on my datacenter
- I must comply to a set of policies when executing steps
- I want to execute an action (for simplicity) but it doesn't provide all the information I need

Customizations!

- It's hard to personalize a world-wide multi-tenant tool...
- GitHub has some extensibility points, not only API based...

There are two main options:

- Use PowerShell, Bash, CMD or whatever to script your requirements
- .NET tools to the rescue!
- Create a custom GitHub Action!

Should I use scripts?

Pros

- You can write any code you need in the way you need to comply to requirements (i.e. PowerShell on Windows, Bash on Linux)
- Can re-use existing code, if any
- There is plenty of documentation available for any command

Cons

- Parameters must be validated at runtime (no compile-time checking)
- May be hard to get started if you are not skilled enough
- Can be tricky if the same script has to run against different OS

.NET tools

.NET tools

A .NET tool is a special NuGet package that contains an entire console application written in .NET

Why a .NET tool?

- Fits perfectly in between the scripts and task extensions
- You can choose your own language that is part of the .NET family
- No UI and no complexity in getting started
- Tasks can work on any OS given .NET 6 is cross-platform (since .NET Core)
- Takes advantage of CI/CD system for ALM and versioning as it's a "classic" console app
- Azure DevOps? Yes. GitHub? Yes. What about Jenkins? YES!

How to get started?

On the authoring side:

1. Write a console app
2. Extend it to be a .NET tool
3. Distribute it on NuGet (or similar)

On the consumer side:

1. Use [dotnet search](#) or [ToolGet](#) to search for packages (optional)
2. Install the tool using `dotnet tool install -g {{ package-name }}`
3. Invoke the tool using `{{ package-name }}` like a "classic" script

Use the task

GitHub

```
# Install the .NET tool
- name: Install tool
  run: dotnet tool install --tool-path tools Hello --version 1.0.0

# When the tool is installed, then we can use it
- name: Print hello
  run: ./tools/hello-world-aspitalia $speaker
  env:
    SPEAKER: "Matteo"
```

Azure DevOps

```
# Install the .NET tool
- script: dotnet tool install --tool-path tools Hello --version 1.0.0
  displayName: 'Install tool'
  env:
    DOTNET_SKIP_FIRST_TIME_EXPERIENCE: true
    DOTNET_CLI_TELEMETRY_OPTOUT: true

# When the tool is installed, then we can use it
- pwsh: ./tools/hello-world-aspitalia ${{ parameters.speaker }}
  displayName: 'Print hello'
```

Creating the action

Customizing actions

It's not really "custom", as you can see from the Marketplace or from any action you may be already using

It's a way to create your own set of instructions to be executed in a different way compared to scripting

You can create an entire lifecycle management for your actions, that is particularly useful when complexity starts raising up

Learning curve is quite flat given what you need to get started is just a repo in GitHub (and an account)

Think ahead about releasing an action

Especially needed when thinking of releasing publicly, because you want to control releases and avoid creating breaking changes for your customers

Use a README.md to explain how the action works

Consider applying a standard branching strategy (i.e. main as default branch, then conventional commits...)

Consider applying a standard versioning strategy (i.e. SemVer)

```
steps:  
- uses: actions/javascript-action@v1.0.0
```

Types of action

Several types of actions that can be used depending on knowledge, customization needed, requirements on runners...

Independently from the type, we need to setup an *action.yml* (or *.yaml*) that defines the entry point of the custom action, structure, input and output parameters and so on...

Type	Can execute on	Self-hosted support	Notes
Docker container	Linux	✓	
JavaScript	Linux, macOS, Windows	✓	
Composite Actions	Linux, macOS, Windows	✓	Can combine multiple workflows in one

JavaScript

To ensure your JavaScript actions are compatible with runners operating systems, the JavaScript code you write should be pure JavaScript

JavaScript actions can run directly on a runner machine and can use binaries that already exist in the virtual environment

Separate action code from the environment used to run the code

If you're developing a Node.js project, the [GitHub Actions Toolkit](#) provides packages that you can use in your project to speed up development

Fastest and most flexible option

Docker containers

More consistent and reliable unit of work because the consumer of the action does not need to worry about the tools or dependencies (as these are already packaged in the Docker container)

Docker based actions can only execute on runners with a Linux operating system

Because of the latency to build and retrieve the container, Docker container actions are slower than JavaScript actions

Dockerfile

```
# Container image that runs your code  
FROM alpine:3.10
```

```
# Copies your code file from your action repository to the filesystem  
path `/` of the container  
COPY entrypoint.sh /entrypoint.sh
```

```
# Code file to execute when the docker container starts up  
(`entrypoint.sh`)  
ENTRYPOINT ["/entrypoint.sh"]
```

entrypoint.sh

```
#!/bin/sh -l  
  
echo "Hello $1"  
time=$(date)  
echo "::set-output name=time::$time"
```

action.yml

```
name: 'Hello World'
description: 'Greet someone and record the time'
inputs:
  who-to-greet: # id of input
    description: 'Who to greet'
    required: true
    default: 'World'
outputs:
  time: # id of output
    description: 'The time we greeted you'
runs:
  using: 'docker'
  image: 'Dockerfile'
  args:
    - ${{ inputs.who-to-greet }}
```

Action and outputs

on: [push]

jobs:

hello_world_job:

runs-on: ubuntu-latest

name: A job to say hello

steps:

To use this repository's private action,
you must check out the repository

- name: Checkout

uses: actions/checkout@v2

- name: Hello world action step

uses: ./ # Uses an action in the root directory

id: hello

with:

who-to-greet: 'Mona the Octocat'

Use the output from the `hello` step

- name: Get the output time

run: echo "The time was \${ steps.hello.outputs.time }"

Injected at runtime!

A job to say hello
succeeded 16 minutes ago in 6s

Set up job 3s

Build actions/hello-world-docker-action@v1 3s

Hello world action step 0s

```
1 ▶ Run actions/hello-world-docker-action@v1
4 /usr/bin/docker run --name f57a5df354401b9ed3bbcd98f1958_bdd3b3 --label 179394 --workdir /github/workspace --rm
-e INPUT_WHO-TO-GREET -e HOME -e GITHUB_JOB -e GITHUB_REF -e GITHUB_SHA -e GITHUB_REPOSITORY -e
GITHUB_REPOSITORY_OWNER -e GITHUB_RUN_ID -e GITHUB_RUN_NUMBER -e GITHUB_RETENTION_DAYS -e GITHUB_ACTOR -e
GITHUB_WORKFLOW -e GITHUB_HEAD_REF -e GITHUB_BASE_REF -e GITHUB_EVENT_NAME -e GITHUB_SERVER_URL -e GITHUB_API_URL
-e GITHUB_GRAPHQL_URL -e GITHUB_WORKSPACE -e GITHUB_ACTION -e GITHUB_EVENT_PATH -e GITHUB_ACTION_REPOSITORY -e
GITHUB_ACTION_REF -e GITHUB_PATH -e GITHUB_ENV -e RUNNER_OS -e RUNNER_TOOL_CACHE -e RUNNER_TEMP -e
RUNNER_WORKSPACE -e ACTIONS_RUNTIME_URL -e ACTIONS_RUNTIME_TOKEN -e ACTIONS_CACHE_URL -e GITHUB_ACTIONS=true -e
CI=true -v "/var/run/docker.sock":"/var/run/docker.sock" -v "/home/runner/work/_temp/_github_home":"/github/home"
-v "/home/runner/work/_temp/_github_workflow":"/github/workflow" -v
"/home/runner/work/_temp/_runner_file_commands":"/github/file_commands" -v "/home/runner/work/hello-world-docker-
action/hello-world-docker-action":"/github/workspace" 179394:59f57a5df354401b9ed3bbcd98f1958 "Mona the Octocat"
5 Hello Mona the Octocat
```

Get the output time 0s

```
1 ▶ Run echo "The time was Mon Nov 30 18:27:49 UTC 2020"
4 The time was Mon Nov 30 18:27:49 UTC 2020
```

Complete job 0s

Limits

Some Docker instructions interact with GitHub Actions, and an action's metadata file can override some Docker instructions

Docker actions must run with the default Docker user (*root*). **Do not** use the *USER* instruction in your Dockerfile, because you won't be able to access the *GITHUB_WORKSPACE*

It's recommended to use Docker images based on the Debian operating system

GitHub sets the working directory path in the *GITHUB_WORKSPACE* environment variable so it's recommended to not use the *WORKDIR* instruction in Dockerfile

Entrypoint in action's metadata file overrides *ENTRYPOINT* defined in Dockerfile

Args in action's metadata file overrides *CMD* defined in Dockerfile

Run images from external registries

```
runs:  
  using: 'docker'  
  image: 'Dockerfile'
```

```
runs:  
  using: 'docker'  
  image: 'docker://debian:stretch-slim'
```

What about .NET 6? 😊

Well... it's Docker, isn't it? 😊

Tags	Dockerfile	OS Version	Last Modified
6.0.100-bullseye-slim-amd64, 6.0-bullseye-slim-amd64, 6.0.100-bullseye-slim, 6.0-bullseye-slim, 6.0.100, 6.0, latest	Dockerfile	Debian 11	11/17/2021
6.0.100-alpine3.14-amd64, 6.0-alpine3.14-amd64, 6.0-alpine-amd64, 6.0.100-alpine3.14, 6.0-alpine3.14, 6.0-alpine	Dockerfile	Alpine 3.14	11/12/2021
6.0.100-focal-amd64, 6.0-focal-amd64, 6.0.100-focal, 6.0-focal	Dockerfile	Ubuntu 20.04	11/08/2021
5.0.403-bullseye-slim-amd64, 5.0-bullseye-slim-amd64, 5.0.403-bullseye-slim, 5.0-bullseye-slim	Dockerfile	Debian 11	11/17/2021
5.0.403-buster-slim-amd64, 5.0-buster-slim-amd64, 5.0.403-buster-slim, 5.0-buster-slim, 5.0.403, 5.0	Dockerfile	Debian 10	11/17/2021
5.0.403-alpine3.14-amd64, 5.0-alpine3.14-amd64, 5.0-alpine-amd64, 5.0.403-alpine3.14, 5.0-alpine3.14, 5.0-alpine	Dockerfile	Alpine 3.14	11/12/2021
5.0.403-alpine3.13-amd64, 5.0-alpine3.13-amd64, 5.0.403-alpine3.13, 5.0-alpine3.13	Dockerfile	Alpine 3.13	11/12/2021
5.0.403-focal-amd64, 5.0-focal-amd64, 5.0.403-focal, 5.0-focal	Dockerfile	Ubuntu 20.04	11/08/2021
3.1.415-bullseye, 3.1-bullseye	Dockerfile	Debian 11	11/17/2021
3.1.415-buster, 3.1-buster, 3.1.415, 3.1	Dockerfile	Debian 10	11/17/2021
3.1.415-alpine3.14, 3.1-alpine3.14, 3.1-alpine	Dockerfile	Alpine 3.14	11/12/2021
3.1.415-alpine3.13, 3.1-alpine3.13	Dockerfile	Alpine 3.13	11/12/2021
3.1.415-focal, 3.1-focal	Dockerfile	Ubuntu 20.04	11/08/2021
3.1.415-bionic, 3.1-bionic	Dockerfile	Ubuntu 18.04	11/08/2021

Benefits

Code reuse (when I have a pre-existing console app written in .NET)

Knowledge, if my team already works with .NET and/or is familiar with Docker containers

Can reuse all the native commands from GitHub to integrate with console outputs, enumerate error messages, stop a workflow, write error messages

Demo

C# Code Analyzer

.NET Conference
Italia 2021



.NET

GitHub Apps

It can be any application that reacts to some events occurring in GitHub, so it's can be an extension of an action

It can be either Node-based or also Blazor/ASP.NET 6 or any other tech stack as long as it can receive any request coming as webhook

```
> POST /payload HTTP/2
> Host: localhost:4567
> X-GitHub-Delivery: 72d3162e-cc78-11e3-81ab-4c9367dc0958
> X-Hub-Signature: sha1=7d38cdd689735b008b3c702edd92eea23791c5f6
> X-Hub-Signature-256: sha256=d57c68ca6f92289e6987922ff26938930f6e3c
> User-Agent: GitHub-Hookshot/044aadd
> Content-Type: application/json
> Content-Length: 6615
> X-GitHub-Event: issues

> {
>   "action": "opened",
>   "issue": {
>     "url": "https://api.github.com/repos/octocat/Hello-World/issues/1347",
>     "number": 1347,
>     ...
>   },
>   "repository": {
>     "id": 1296269,
>     "full_name": "octocat/Hello-World",
>     "owner": {
>       "login": "octocat",
>       "id": 1,
>       ...
>     },
>     ...
>   },
>   "sender": {
>     "login": "octocat",
>     "id": 1,
>     ...
>   }
> }
```

GitHub Actions vs GitHub Apps

Actions

- Provide automation that can perform continuous integration and continuous deployment.
- Can run directly on runner machines or in Docker containers.
- Can include access to a clone of your repository, enabling deployment and publishing tools, code formatters, and command line tools to access your code.
- Don't require you to deploy code or serve an app.
- Have a simple interface to create and use secrets, which enables actions to interact with third-party services without needing to store the credentials of the person using the action.

Apps

- Run persistently and can react to events quickly.
- Work great when persistent data is needed.
- Work best with API requests that aren't time consuming.
- Run on a server or compute infrastructure that you provide.

Summary

Different ways to execute GitHub Actions

Different ways and complexities involved when running either scripts, .NET tools or creating custom actions

Everything depends on time and on the business needs, as always 😊

[Demo available in GitHub](#)

@xtumiox
matteot@aspitalia.com

Slide e materiale su
<https://aspit.co/netconfit-21>

.NET Conference
Italia 2021

.NET