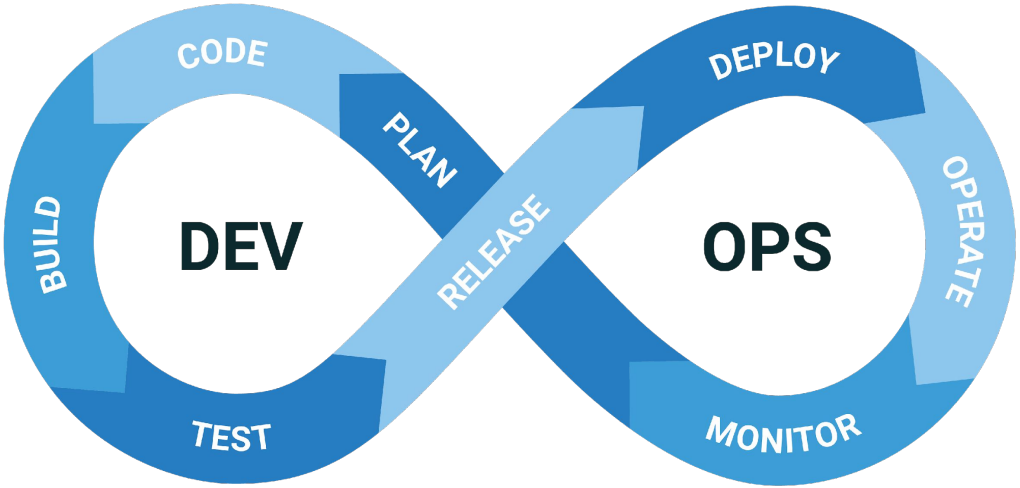
Continuous integration (CI) / Continuous delivery (CD)

Mads Nyborg
Business Academy Copenhagen

Continuous Integration / Continuous Delivery pipeline

- Code
  - The developer writes code locally.
  - The code is stored in a version control system (locally: git - remotely: GitHub, GitLab, or Bitbucket).
- Build
  - The code is compiled or packaged if needed (e.g., Java to .class, TypeScript to JavaScript).
  - Dependencies are installed (e.g., npm install, mvn install).
- Test
  - Unit tests (eg. Jest), integration tests, or end-to-end tests are run automatically.
  - The goal is to catch bugs early.
- Release
  - Approved code is prepared for deployment.
  - This can mean creating a release build or generating a Docker image.
- Deploy
  - The code is rolled out to staging or production.
  - Can be automatic (CD) or require manual approval.
- Operate
  - The application runs in production.
  - Monitoring and logging detect issues.
- Monitor
  - Performance, stability, and user behavior are tracked.
  - Reports or alerts can trigger new backlog items.

DevOps culture



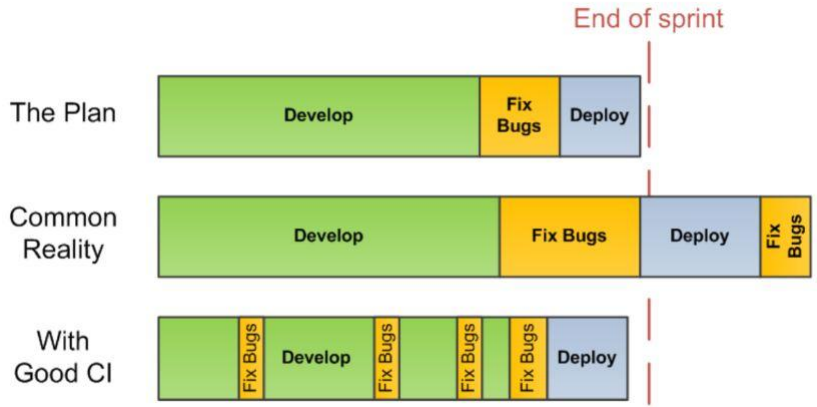
DevOps is a cultural and organizational approach that fosters collaboration between development and operations teams
CI/CD (continuous integration and continuous delivery) is a set of practices for automating the integration and deployment of code changes

Why CI/CD ?

- Frequent changes -> less integration problems
- Bugs are detected early -> saves money
- Avoid last minute chaos
- Transparency to all
- Testing on production-like environment
- Easy to rollback in case of any issue
- Enforce of automation culture
- Enforce DevOps culture
- Make the developers accountable and take ownership

Integration early

Continuous integration involves integration early and often, in order to avoid “Integration hell”

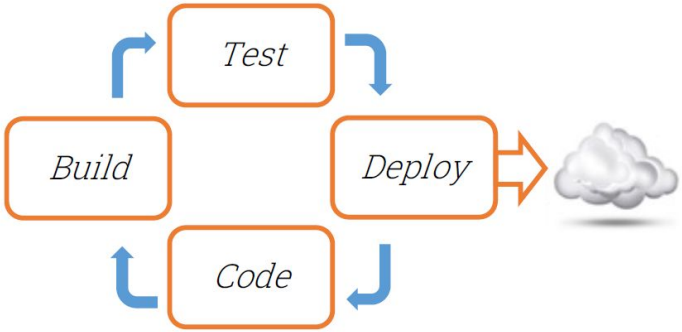


CI Principles

Automate the build (single command), Automate all. Minimize human error	The entire build process (compiling, packaging, etc.) should be runnable with one command — no manual steps.
Build is self-testable Ensure code integrity Keep the branch stable	The build should automatically run tests to verify the integrity of the code.
Baseline branch is open consistently Keep the branch stable	The main branch (e.g., main or master) should always be in a deployable state.
Every commit should be built Detect issues early	Every commit triggers a build (and ideally tests).
Build should be fast Increase developer produuctivity	Feedback should be quick so developers know within minutes if something failed.
Test Env is clone of Production (as much as we can) Reduce deployment riscaks	The test environment should mimic production as closely as possible (OS, database, configurations).
Everyone can see the status – Transparency Encourage shared responsibility	Build and test results should be visible to the entire team.

Continuous Delivery & Deployment

Continuous Delivery (CD) is a software development discipline where you build software in such a way that the software **can be released to production at any time**.



Continuous Deployment means that every change goes through the pipeline and automatically gets put into production, resulting in many production deployments every day.

CD is not only in Computer Software

Tesla Model S gets firmware updates on regular basis for both UI and major elements (suspension, acceleration and more)

Github actions

CI and GitHub Actions

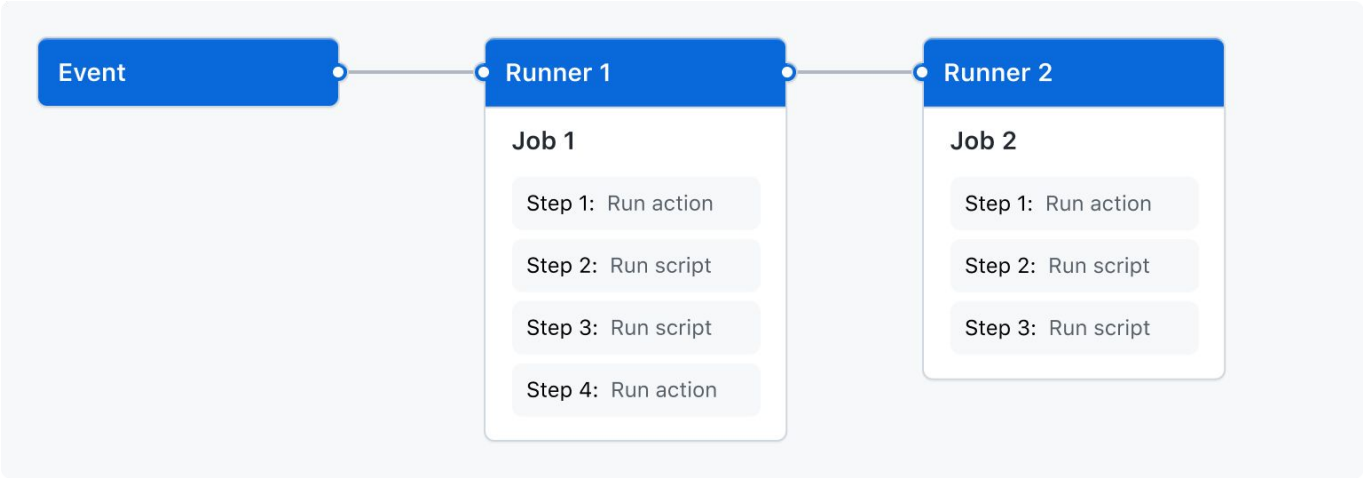
- Use GitHub Actions to create workflows that can **build the code** in your repository and **run your tests.**
- Workflows can run on **GitHub-hosted virtual machines**.
- Configure a CI workflow to run when a **GitHub event** occurs (e.g. when new code is pushed to your repository).
- GitHub runs your CI tests and provides the results of each test in the pull request, so you can see whether the change in your branch introduces an error.
- When all CI tests in a workflow pass, the changes you pushed are ready to be reviewed by a team member or merged.

GitHub Actions - workflow

- Understanding GitHub Actions

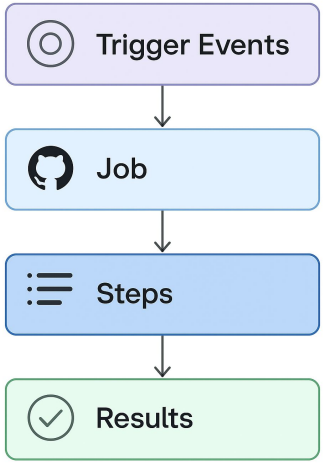
specified in .yml file

must be stored in a subfolder named /.github/workflows



GitHub Actions - workflow .yml file

- Workflows
  - configurable automated process that will run one or more jobs
  - defined by a YAML file checked in to your repository
  - run when triggered by an event in your repository e.g. push or manually, or at a defined schedule.



GitHub Actions

- Job
  - set of steps in a workflow that is executed on the same runner.
  - Each step is either a shell script that will be executed, or an action that will be run.
  - Steps are executed in order and are dependent on each other.
  - Since each step is executed on the same runner, you can share data from one step to another. E.g., you can have a step that builds your application followed by a step that tests the application that was built.

GitHub Actions

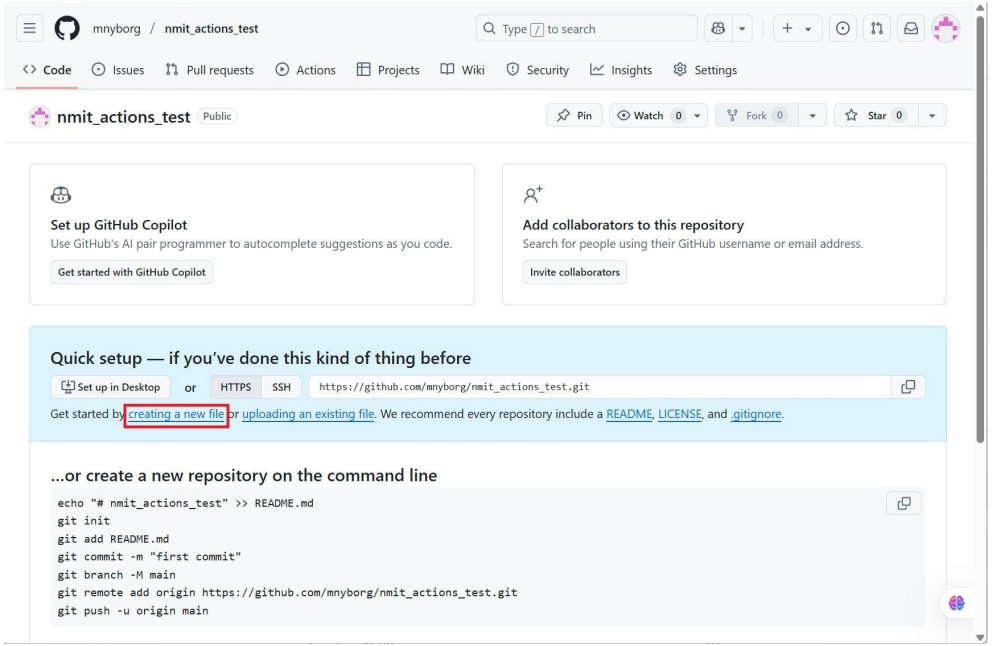
- Runner
  - a server that runs your workflows when they're triggered.
  - Each runner can run a single job at a time.
  - GitHub provides Ubuntu Linux, Microsoft Windows, and macOS runners to run your workflows.
  - Each workflow run executes in a fresh, newly-provisioned virtual machine.

Workflow file - example

```
name: Hello and Goodbye Workflow

on:
  push:
    branches:
      - main           # Only run when pushing to the main branch
  workflow_dispatch:   # Manual trigger from GitHub Actions UI

jobs:
  hello:
    runs-on: ubuntu-latest # spin up a virtual ubuntu server on GitHub
    steps:
      - name: Say Hello
        run: echo "Hello world"

  goodbye:
    runs-on: ubuntu-latest
    steps:
      - name: Say Goodbye
        run: echo "Goodbye"
```

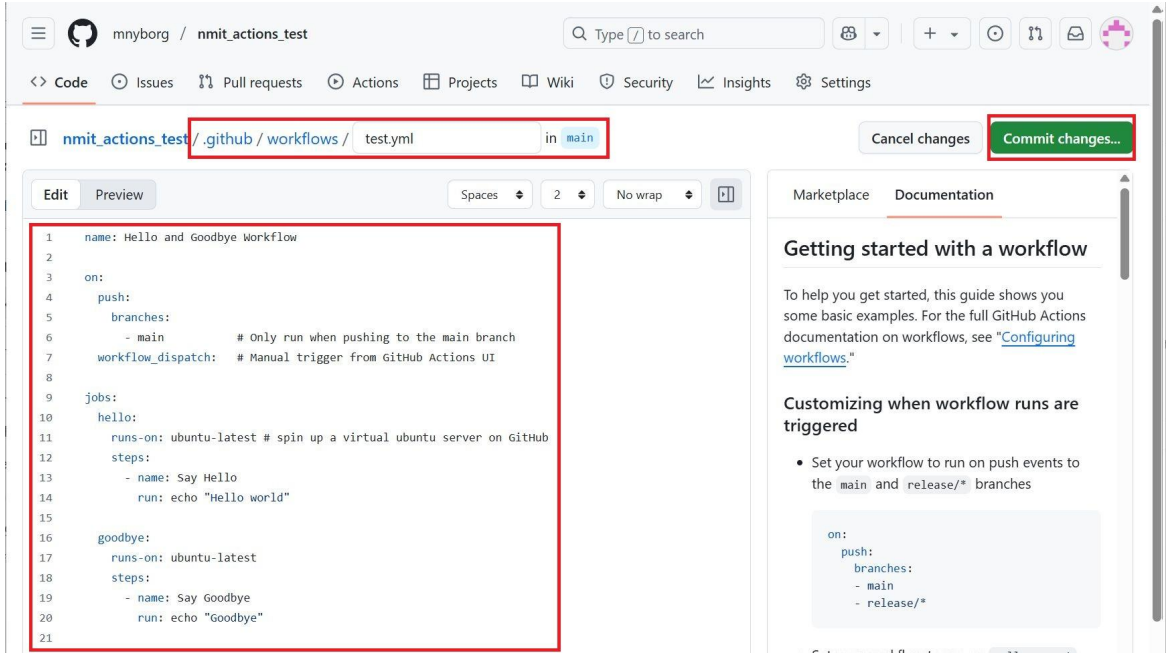
Now let’s make the example …

Make a repository and click “creating a file”



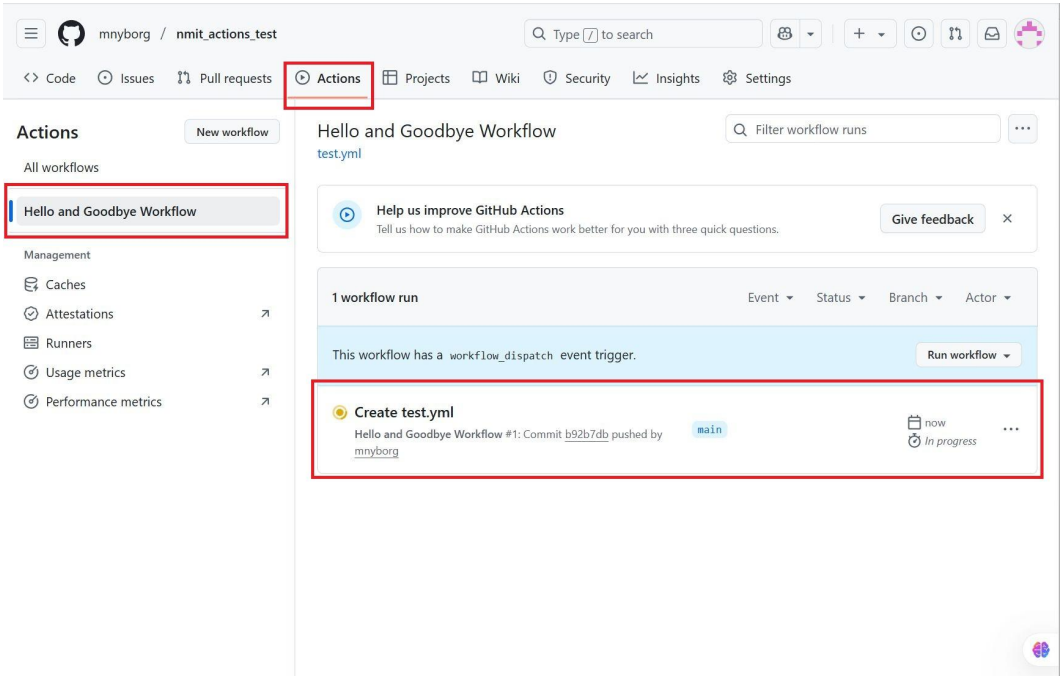
Now let’s make the example …

Enter file name. /.github/workflows/test.yml and content. Commit changes



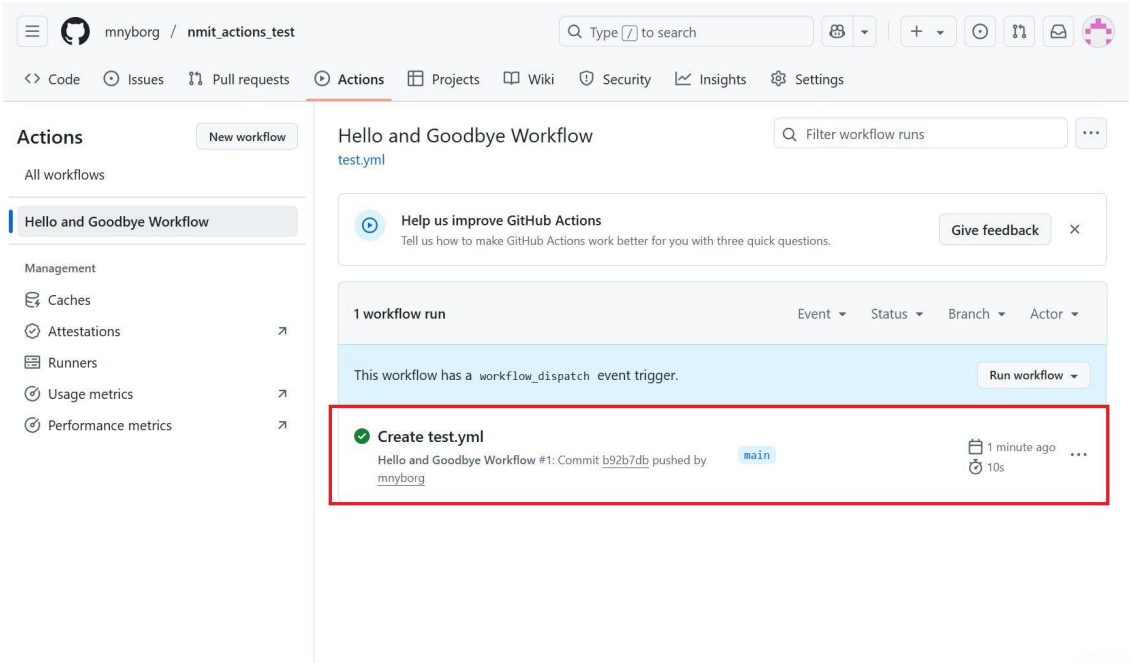
Now let’s make the example …

Observe workflow run



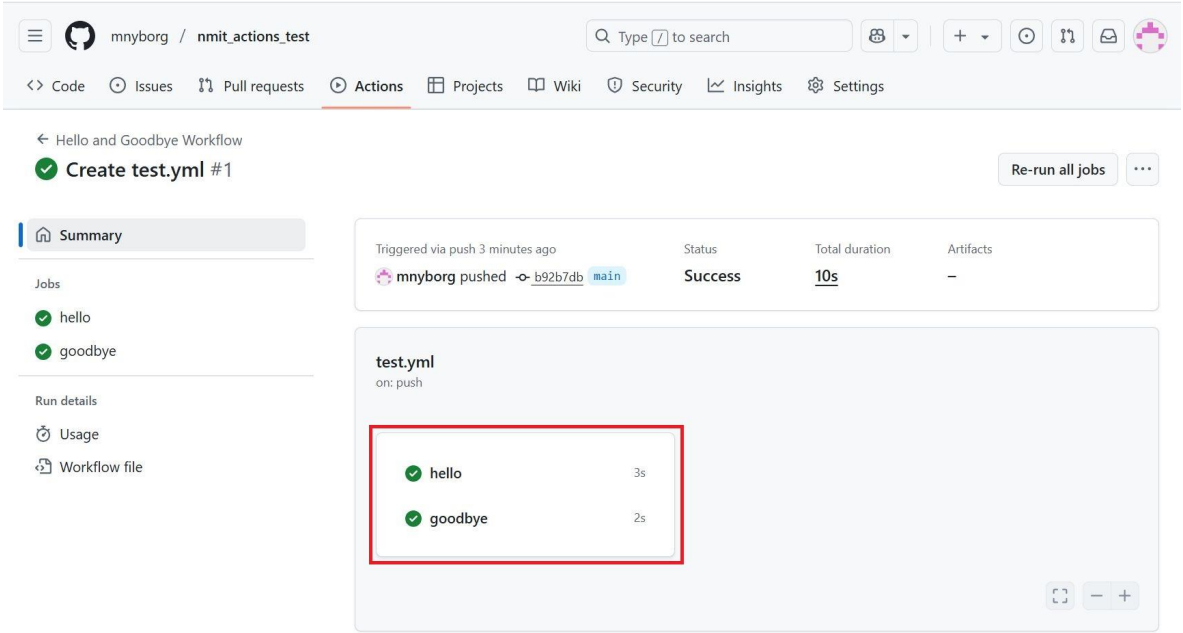
Now let’s make the example …

Observe workflow completes



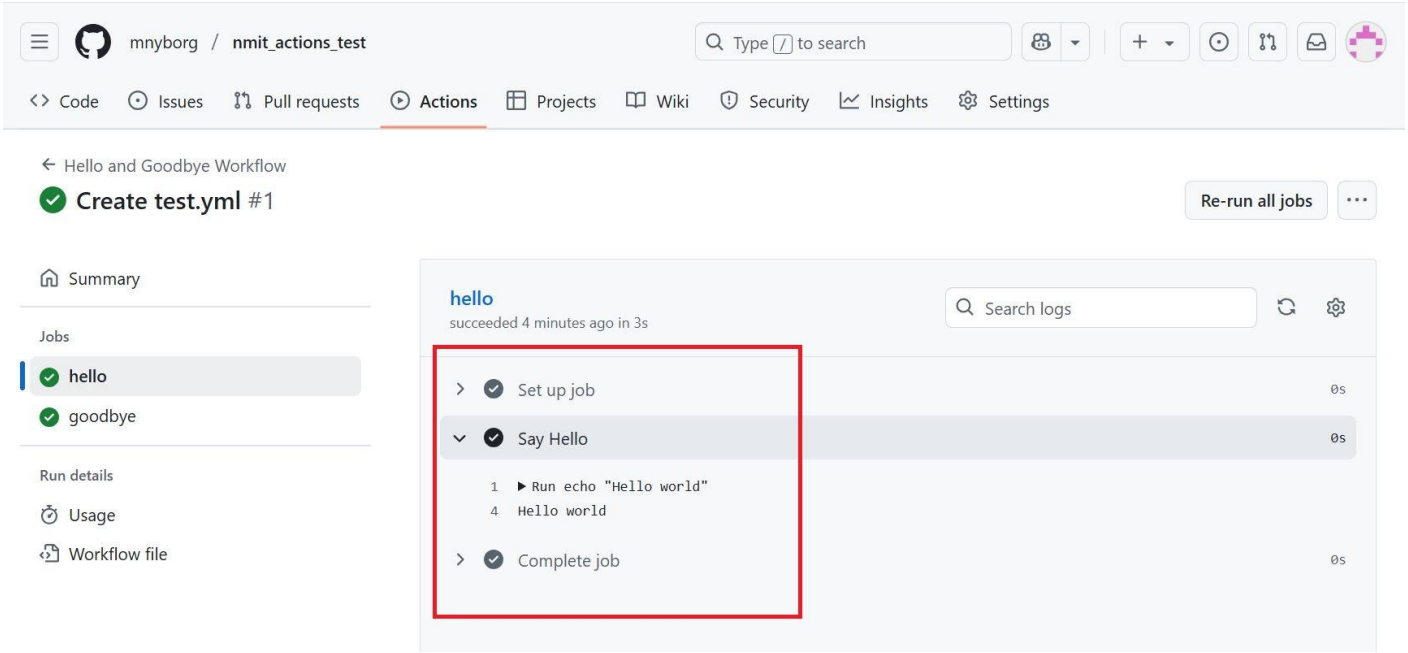
Now let’s make the example …

Observe workflow complete - detailed view



Now let’s make the example …

Observe workflow results

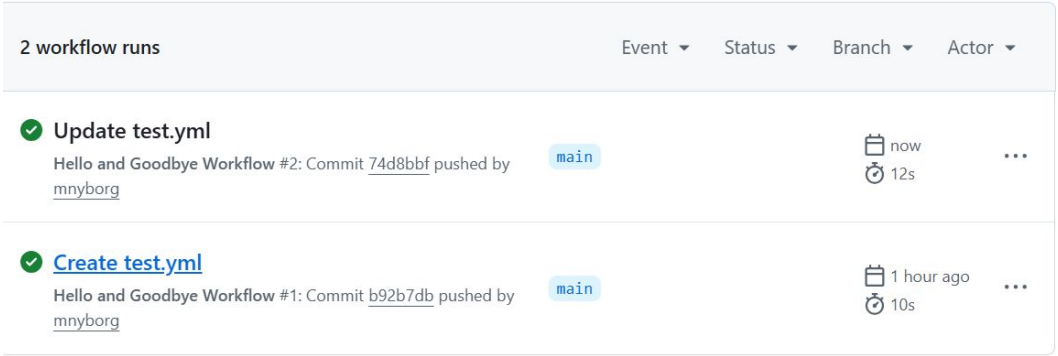


Now let’s make the example …

- Make a change to the output from the echo command in the hello job:

```
steps:
  - name: Say Hello
    run: echo "Hello world 2"
```

- Commit changes, commit message “Update test.yml”
- Observe the workflow runs again due to the push trigger



We now extend the example …

- We want to add two javascript files hello.js and goodbye.js to the repository that produce the same output as before
- We will need to use two premade action scripts from GitHub marketplace:
  - `actions/checkout` - checks out the content of the repo to the runner
  - `actions/setup-node` - installs node on the runner

hello.js:

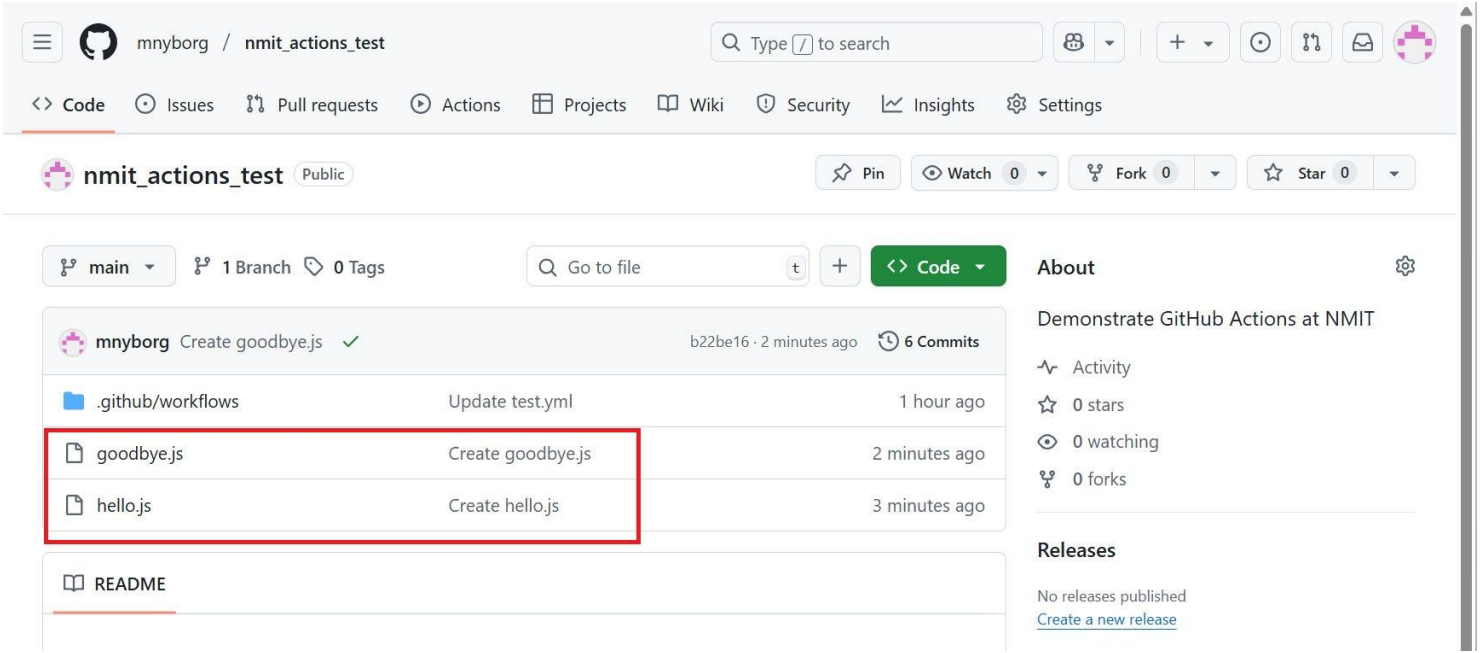
```
console.log("Hello from JavaScript!");
```

goodbye.js:

```
console.log("Goodbye from JavaScript!");
```

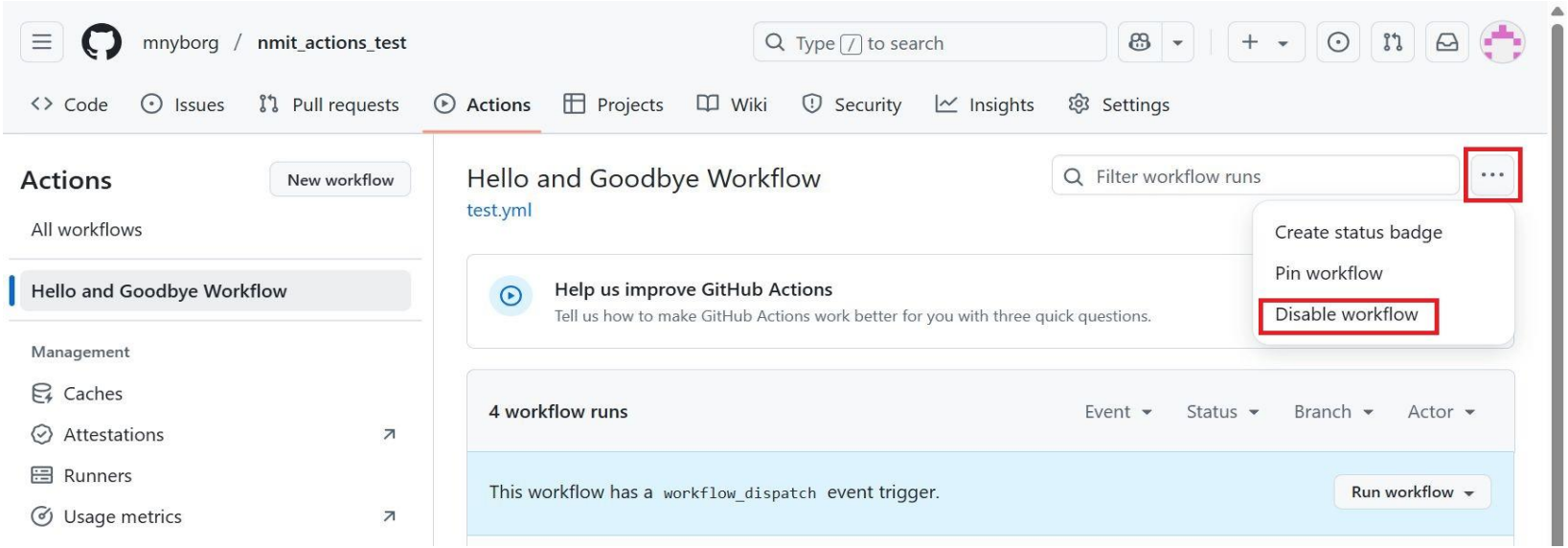
We now extend the example …

hello.js and goodbye.js created in root folder of repository



We now extend the example …

Before we define a new workflow we will disable the original workflow



We now extend the example …

Make a new workflow: test2.yml

```
name: Hello and Goodbye Workflow with javascript

on:
  push:
    branches:
      - main
  workflow_dispatch:

jobs:
  hello:
    runs-on: ubuntu-latest  # Use the latest Ubuntu virtual machine
    steps:
      # Download the repository content to the runner
      - name: Check out repository
        uses: actions/checkout@v4

      # Set up a specific Node.js version
      - name: Set up Node.js
        uses: actions/setup-node@v3
        with:
          node-version: 20

      # Run the hello.js script from the repository
      - name: Run Hello script
        run: node hello.js
```

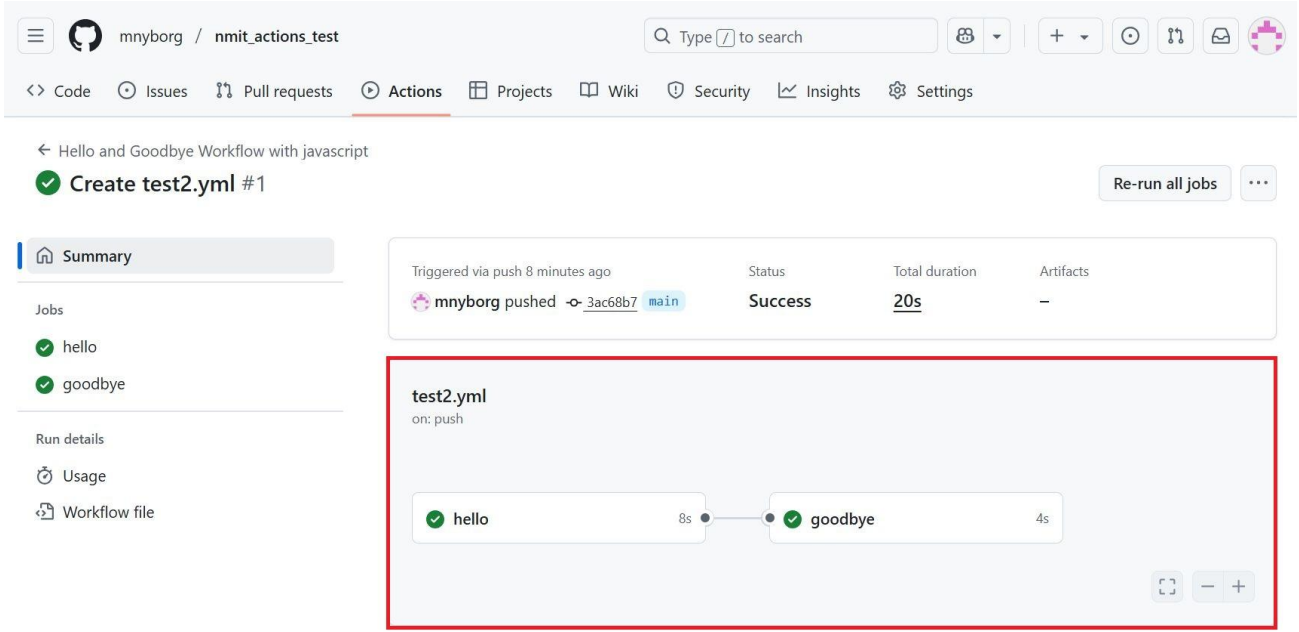
We now extend the example …

Make a new workflow

```
goodbye:
  # This job will only start after the "hello" job has finished successfully
  needs: hello
  runs-on: ubuntu-latest
  steps:
    # Download the repository content again
    - name: Check out repository
      uses: actions/checkout@v4

    # Set up Node.js
    - name: Set up Node.js
      uses: actions/setup-node@v3
      with:
        node-version: 20

    # Run the goodbye.js script from the repository
    - name: Run Goodbye script
      run: node goodbye.js
```

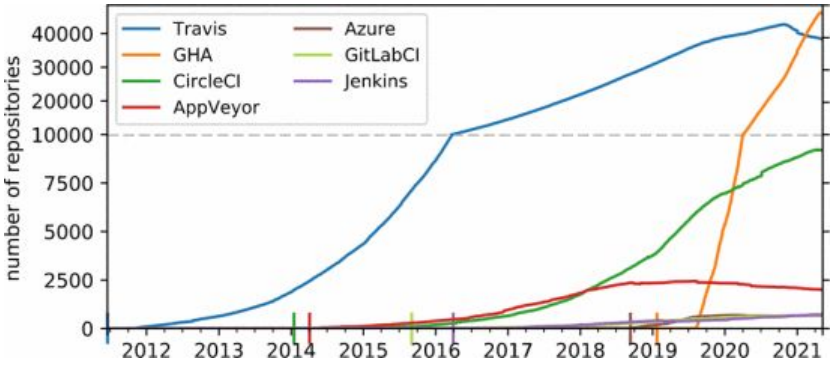
We now extend the example …

Observe the result. Since we have used “needs” the sequence of the jobs is guaranteed



CI platforms and popularity

Number of repositories using a specific CI.



CI	URL	first observed on	repositories #	repositories %	repositories cum. %	usages %	usages cum. %
Travis	http://travis-ci.org	Jun 10, 2011	53,401	58.2%	58.2%	44.9%	44.9%
GHA	http://github.com/features/actions	Jan 23, 2019	46,416	50.6%	90.9%	39.0%	83.9%
CircleCI	http://circleci.com	Jan 15, 2014	11,431	12.4%	98.1%	9.6%	93.5%
AppVeyor	http://www.appveyor.com	Apr 04, 2014	3,553	3.9%	98.3%	3.0%	96.5%
Azure	http://azure.microsoft.com	Sep 11, 2018	1,045	1.1%	98.7%	0.9%	97.3%
GitLab CI	http://docs.gitlab.com/ee/ci	Sep 02, 2015	1,018	1.1%	99.1%	0.9%	98.2%
Jenkins	http://www.jenkins.io	Mar 30, 2016	1,008	1.1%	99.6%	0.8%	99.0%
Others	N/A	Oct 23, 2013	1,138	1.2%	100.0%	1.0%	100.0%

119,033 repositories in total containing configuration files, e.g. .yml files for GHA

Data obtained using libraries.io

Artikel: On the rise and fall of CI services in GitHub

https://ieeexplore.ieee.org/document/9825792