

Table of contents

GitHub Actions	2
First GitHub Action	4
GitHub Action Example Two	7
GitHub Events	14
Job Artifacts	17

GitHub Actions

What is GitHub Actions

GitHub Actions is a workflow automation service offered by GitHub, to automate all kinds of repository related processes and actions.

Repository

Is a bucket that contains the code.

Use Cases for GitHub Actions

Code Deployment (CI/CD)

- Automate code testing, building & deployment.
- Combined, they group methods for automating app development and deployment.
- Code changes are automatically built, tested and merged with existing code (CI)
- After integration, new app or package versions are published automatically.

Code & Repository Management

- Automate Code Reviews, issues management

GitHub Actions Key Element

There are three main key elements in GitHub:

- Workflows
- Jobs
- Steps

Workflows

- Workflows are attached to a GitHub repository
- They Contain one or more jobs
- They are triggered upon Events

Jobs

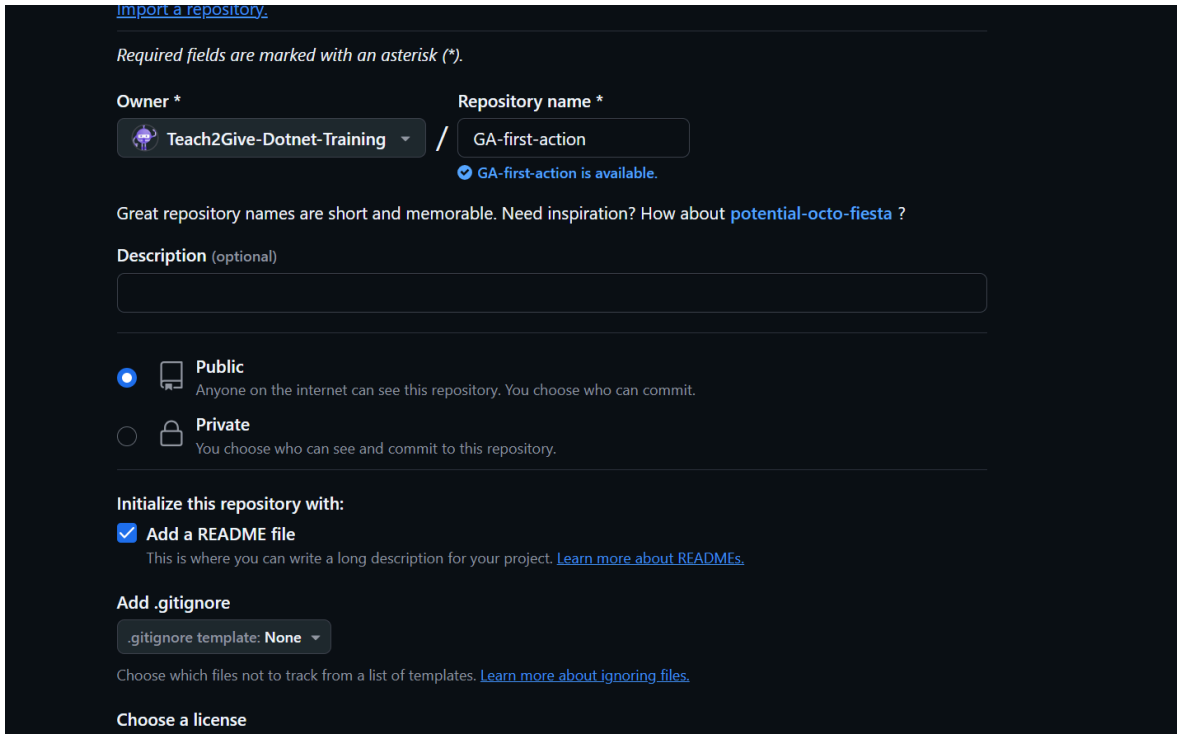
- Jobs define a runner(execution environment)
- Contain one or more steps
- Run in parallel(default) or sequential.
- Can be conditional

Steps

- Execute a shell script or an action.
- Can use custom or third party actions
- Steps are executed in order
- Can be conditional

First GitHub Action

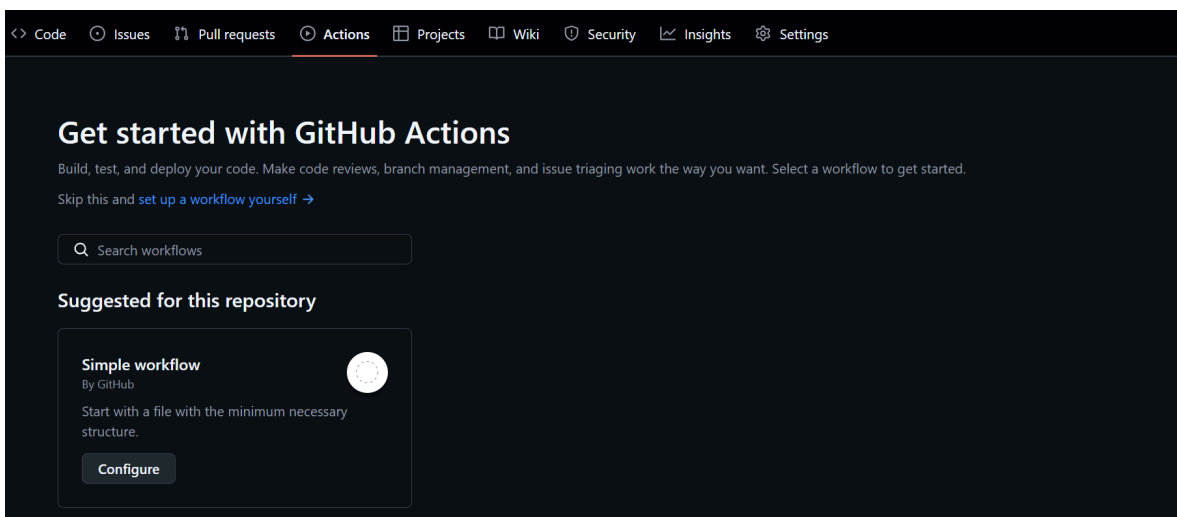
To create a GitHub Action, Let's first Create a repository.



The screenshot shows the 'Create repository' form on GitHub. At the top, there's a link 'import a repository.' and a note 'Required fields are marked with an asterisk (*)'. The 'Owner' field is a dropdown menu showing 'Teach2Give-Dotnet-Training'. The 'Repository name' field contains 'GA-first-action' with a blue checkmark and the text 'GA-first-action is available.' below it. A hint text says 'Great repository names are short and memorable. Need inspiration? How about potential-octo-fiesta?'. The 'Description (optional)' field is empty. Under 'Visibility', the 'Public' radio button is selected, with the text 'Anyone on the internet can see this repository. You choose who can commit.' Below it, the 'Private' option is unselected, with the text 'You choose who can see and commit to this repository.' Under 'Initialize this repository with:', the 'Add a README file' checkbox is checked, with a link 'Learn more about READMEs.' below it. The 'Add .gitignore' section shows a dropdown menu with 'None' selected, and a link 'Learn more about ignoring files.' below it. At the bottom, there's a 'Choose a license' link.

Create Repository

Then click on the Actions tab> then configure.



Action

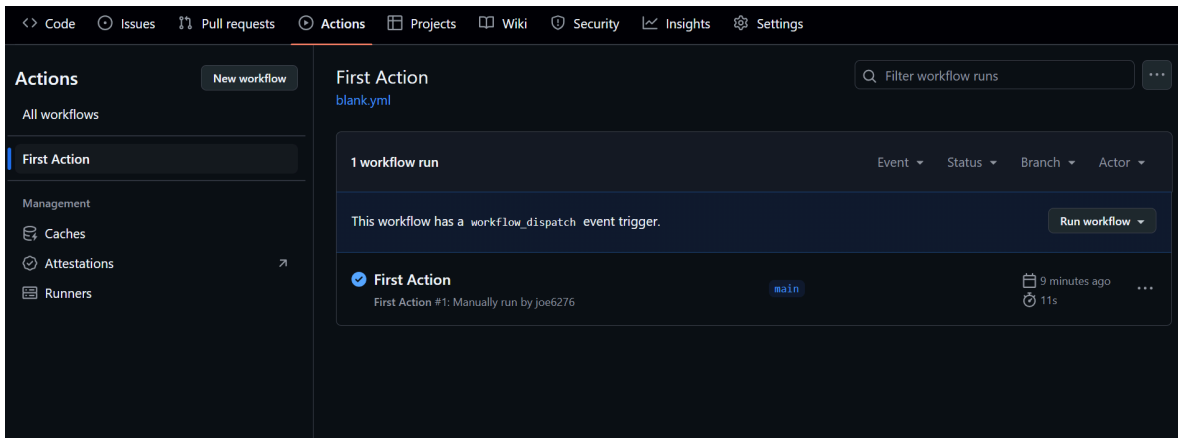
Now delete the steps on the .yaml file and paste the following:

```
name: First Action
# Above is the name of the workflow
on: workflow_dispatch
# This states the event that will trigger the workflow
# in this case "workflow_dispatch" means it will be manually triggered.
jobs:
  ## Now list the job to be executed
  first-job:
    # Give a name (depend on you)
    runs-on: ubuntu-latest
    # This is the virtual machine that will run the workflow
    steps:
      # Now the steps that will be taken
      - name: First hello World
        run: echo " Hello World "
        # Here we give a name, to uniquely identify the step
        # Then we use run to run a shell Command
      - name: Say GoodBye
        run: echo " GoodBye!"
        # Another step
```

Now commit the changes.

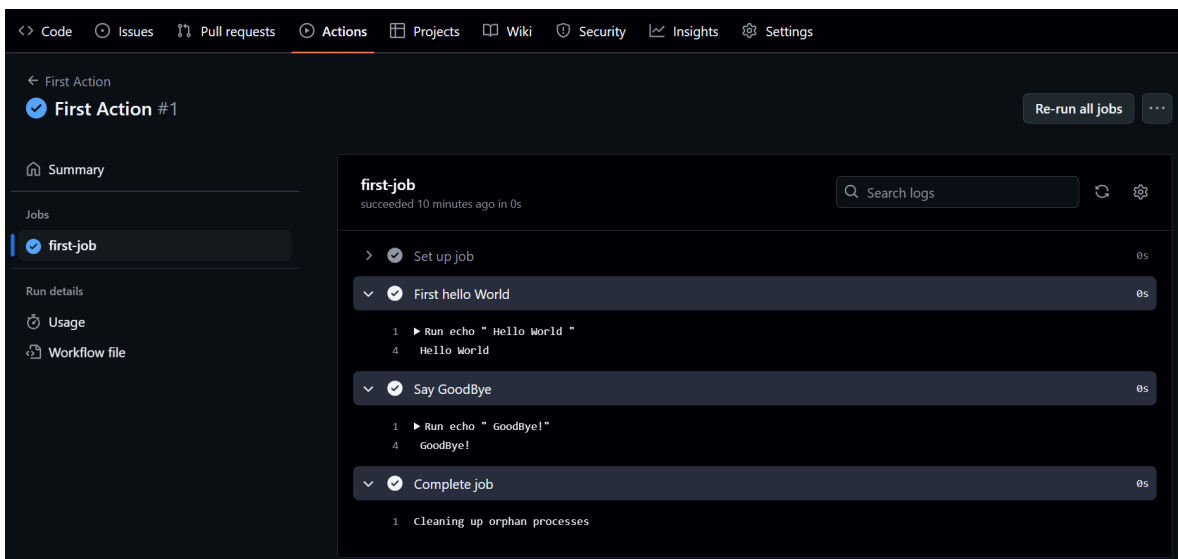
Run the workflow

Under actions > All Workflows You will find the 'First Action' work flow:



First Workflow

Click on run workflow.



Run Workflow

GitHub Action Example Two

Find the project Here: [https://github.com/Teach2Give-Dotnet-](https://github.com/Teach2Give-Dotnet-Training/GA_second_Action)

Training/GA_second_Action This is a simple React App with some test files, the goal will be to write an action that can run the test files.

```
name: Example two
on: push
jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - name: Get the Code from the repository
        uses: actions/checkout@v4

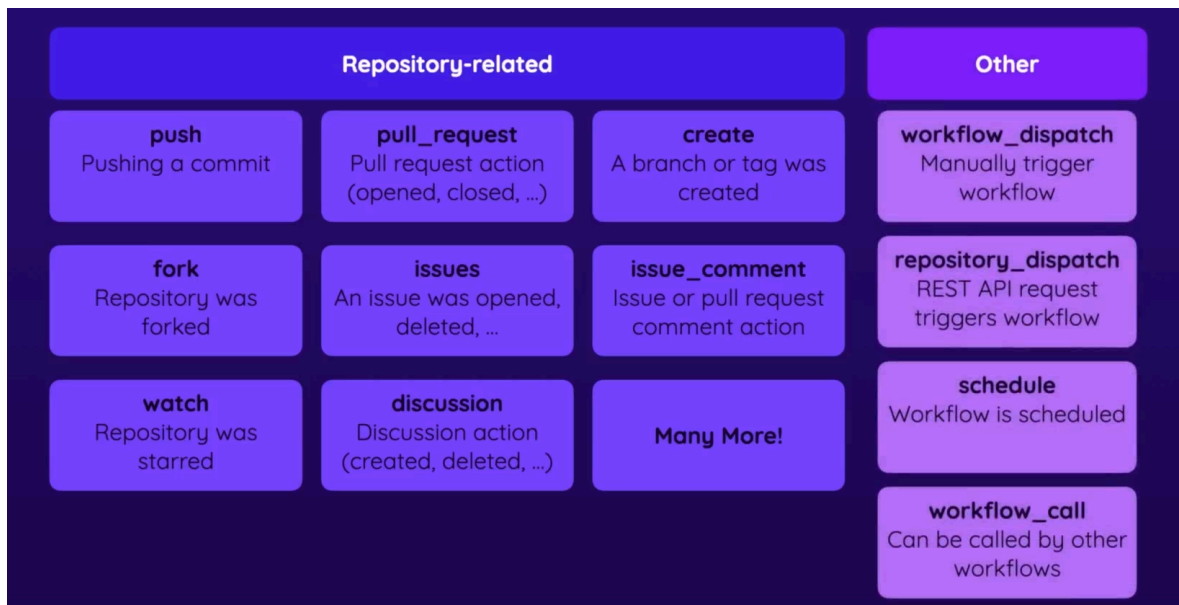
      - name: Install NodeJS
        uses: actions/setup-node@v4
        with:
          node-version: '20'

      - name: Install Dependencies
        run: npm ci

      - name: Run Tests
        run: npm run test
```

We will give it a name. Then run the program whenever we push to the repository.

More on Events



Events

There are so many events that can trigger a workflow. We have repository related events and other events that don't depend on repositories. Events also have variations. To get to know events and variations more you can check at this link

(<https://docs.github.com/en/actions/writing-workflows/choosing-when-your-workflow-runs/events-that-trigger-workflows>)

Action

- A (custom) application that performs a (typically complex) frequently repeated task
- You can build your own actions, but you can also use official or community actions.

An alternative to an action is Command

Command("run")

- A (typically simple) shell command that you define.

```
steps:  
  - name: Get the Code from the repository  
    uses: actions/checkout@v4
```


The ubuntu-latest runner by default does not have the code, the above step will make sure that the code is downloaded from the repository to the virtual machine.

```
- name: Install NodeJS
  uses: actions/setup-node@v4
  with:
    node-version: '20'
```

The above step is optional this is because the runner we are using has Node JS installed already. Here is a List of software installed in it: Link (<https://github.com/actions/runner-images/blob/main/images/ubuntu/Ubuntu2204-Readme.md>)

In case you want to use a different version of Node JS you can now use this action and give a different variation.

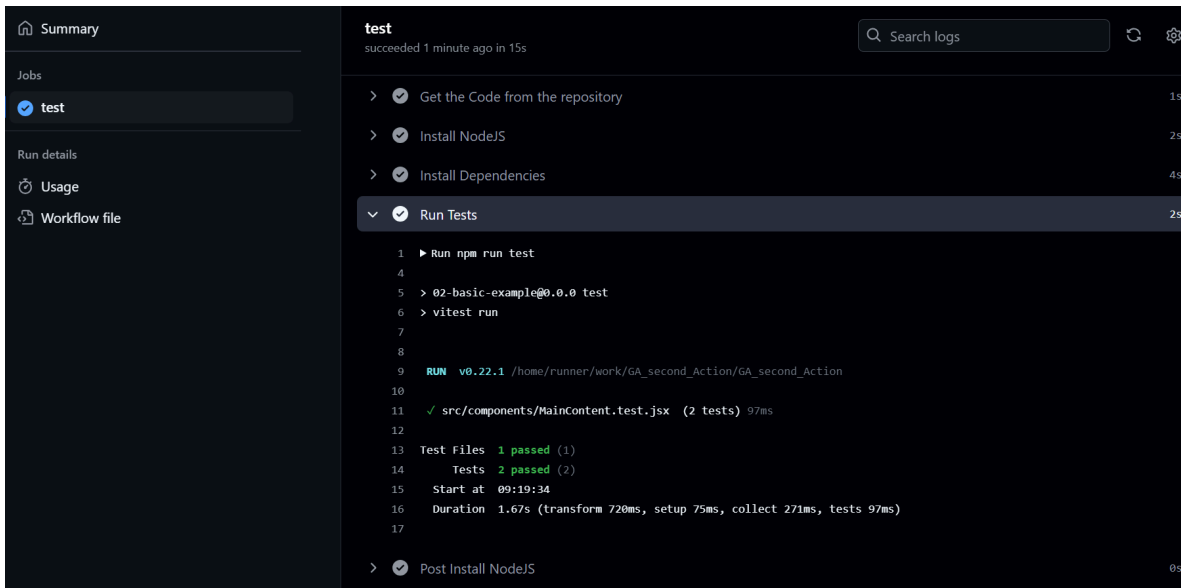
```
- name: Install Dependencies
  run: npm ci
```

This step will make sure that the runner installs the right dependencies needed to run the project.

```
- name: Run Tests
  run: npm run test
```

Now run the command to run the tests. This is also defined in package.json file.

Now push the changes and check your action (Its triggered by a push event)

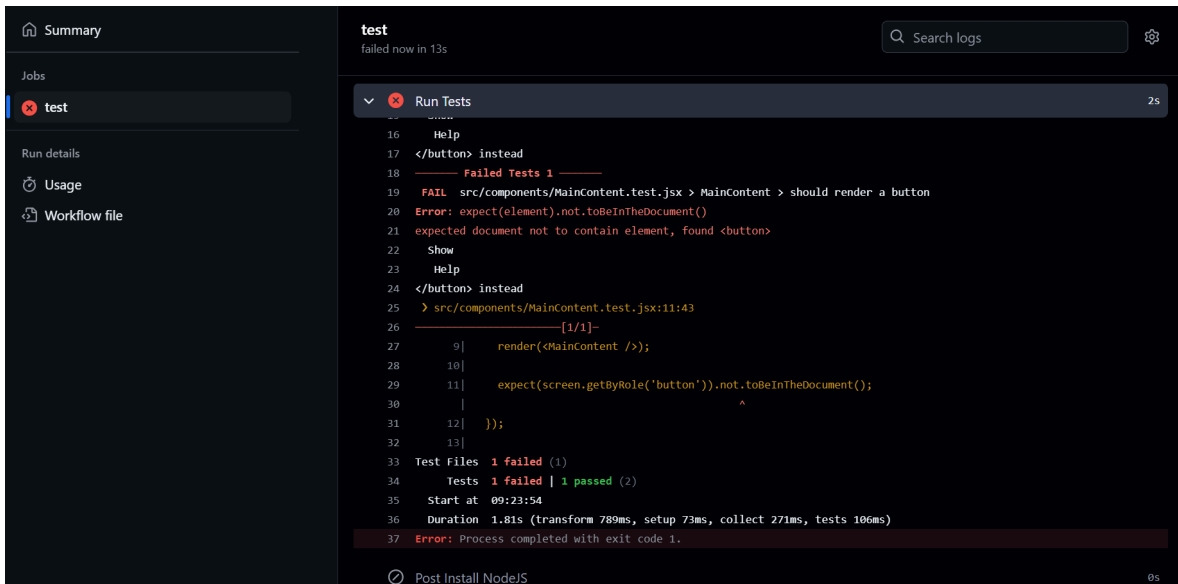


Success

Now let's cause a Bug in the MainContent.test.jsx by negating the test.

```
describe('MainContent', () => {  
  it('should render a button', () => {  
    render(<MainContent />);  
  
    expect(screen.getByRole('button')).not.toBeInTheDocument();  
  });  
});
```

Now push the changes and check the Workflow. Now the workflow Fails:



Failure

Now let's correct the above and add another step then push. The below step will mimic deploying an application

```
name: Example two
on: push
jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - name: Get the Code from the repository
        uses: actions/checkout@v4

      - name: Install NodeJS
        uses: actions/setup-node@v4
        with:
          node-version: '20'

      - name: Install Dependencies
        run: npm ci

      - name: Run Tests
        run: npm run test

    deploy:
```

```

runs-on: ubuntu-latest
steps:
  - name: Get the Code from the repository
    uses: actions/checkout@v4

  - name: Install NodeJS
    uses: actions/setup-node@v4
    with:
      node-version: '20'

  - name: Install Dependencies
    run: npm ci

  - name: Build Projects
    run: npm run build

  - name: Deploy Project
    run: echo " Deploying Project..."

```

Now both steps are successful, but they run in parallel not sequential.

The screenshot shows a GitHub Actions workflow run for 'Running Test Workflow #3'. The workflow is triggered by a push from user 'joe6276' to the '35210e2' commit on the 'master' branch. The status is 'Success'. The total duration is 24s, and the billable time is 2m. The workflow file is 'example2.yml' with the trigger 'on: push'. The workflow consists of two jobs: 'test' (13s) and 'deploy' (11s), which are shown as running in parallel.

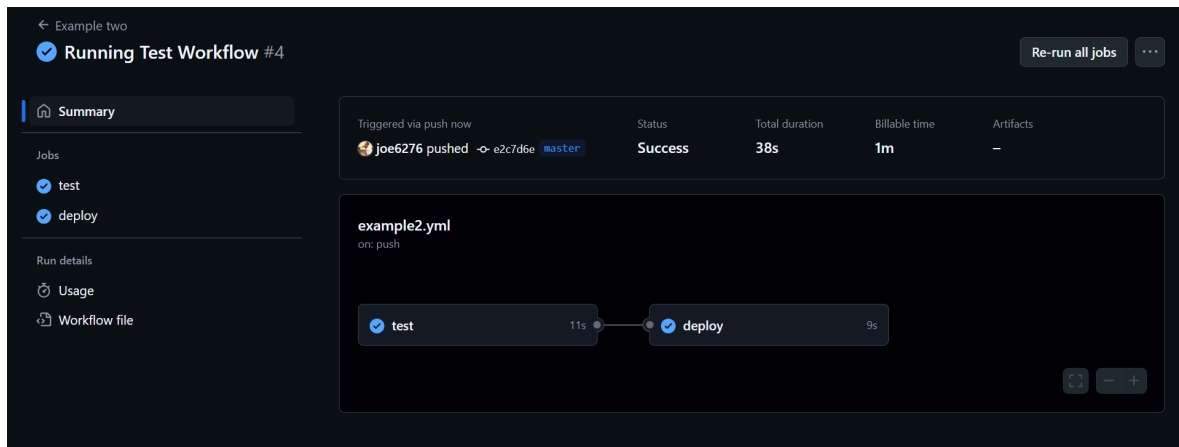
Jobs	Status	Total duration	Billable time	Artifacts
test	Success	13s		
deploy	Success	11s		
Total	Success	24s	2m	-

Steps in Parallel

needs

To make this sequential, in the deploy code add:

```
deploy:
  needs: test
```



sequential

Multi-Events

To trigger workflow based on multiple events

```
on: [push, workflow_dispatch]
```

GitHub Events

Event Activity Types and Filters

some events have Activity Types, others have filters.

Activity Types

More detailed control over when a workflow will be triggered. E.g., pull_request Event Activities include:

- opened
- closed
- edited

```
name: Events Demo 1
on:
  pull_request:
    types: [opened, closed]
  workflow_dispatch:

jobs:
  deploy:
    runs-on: ubuntu-latest
    steps:
      - name: Output event data
        run: echo "${{ toJSON(github.event) }}"
      - name: Get code
        uses: actions/checkout@v3
      - name: Install dependencies
        run: npm ci
      - name: Test code
        run: npm run test
      - name: Build code
        run: npm run build
```

```
- name: Deploy project
  run: echo "Deploying..."
```

```
on:
  pull_request:
    types: [opened, closed]
  workflow_dispatch:
```

This part will make sure that when a pull request is opened or closed, the workflow will be triggered. The workflow can also be triggered manually using the `workflow_dispatch` (note that you have to add the semicolon at the end)

Most events with variation have defaults in case there is no type specified. E.g., for pull-request, the event's activity type is `opened`, `synchronize`, or `reopened`.

Filters

More detailed control over when a workflow will be triggered. E.g., push Event filter based on target branch.

```
push:
  branches:
    - master
  paths-ignore:
    - ".github/workflows/*"
```

The above actions will filter any push that targets the “master” branch. Any change made to any .yml file located in the `.github>workflows` directory.

Cancelling and Skipping Workflow

Cancelling

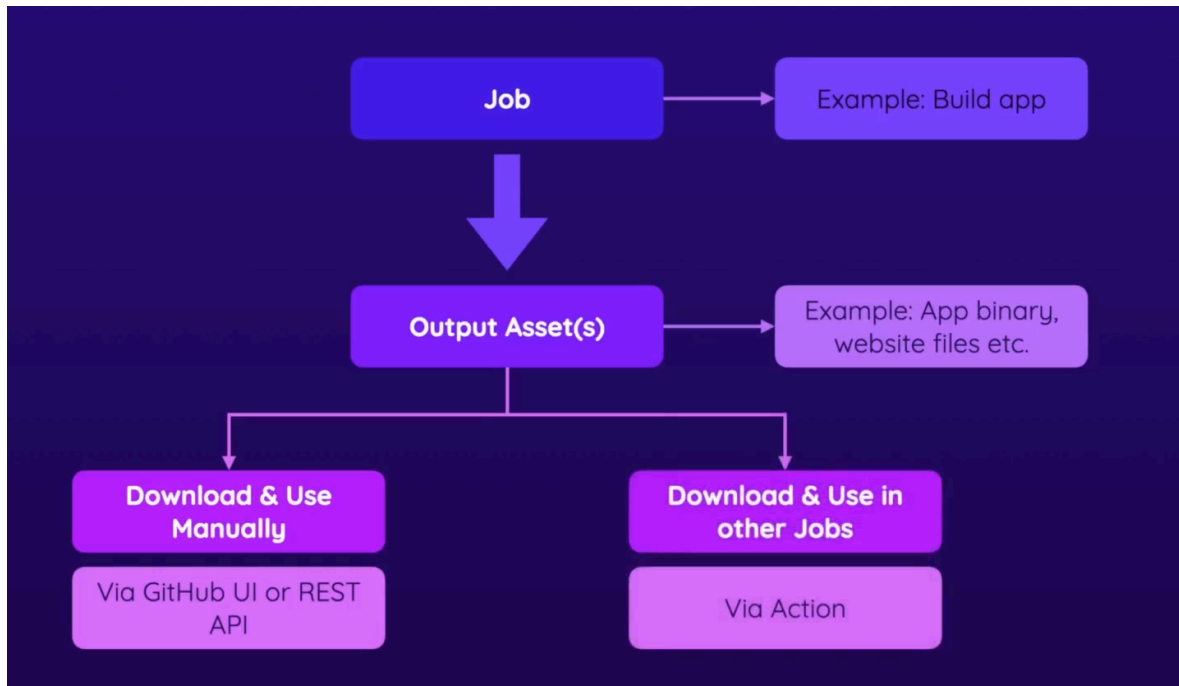
- Workflows get cancelled automatically when jobs fail
- You can manually cancel workflows

Skipping a Workflow

You can Skip via [Skip ci] etc. in the commit message. If the command is part of your commit message, the Workflow won't be executed.

Job Artifacts

Job artifacts are the assets/ output generated by a job.



Build Job

Starting Workflow:

```
name: Deploy website
on:
  push:
    branches:
      - master

jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - name: Get code
        uses: actions/checkout@v3
      - name: Install dependencies
        run: npm ci
```

```

- name: Lint code
  run: npm run lint
- name: Test code
  run: npm run test

build:
  needs: test
  runs-on: ubuntu-latest
  steps:
    - name: Get code
      uses: actions/checkout@v3
    - name: Install dependencies
      run: npm ci
    - name: Build website
      run: npm run build

deploy:
  needs: build
  runs-on: ubuntu-latest
  steps:
    - name: Deploy
      run: echo "Deploying..."

```

The above has three jobs a test, build and deploy. But let's focus on the build job because when the *npm run build* command is executed, it produces a dist file which is the file that is uploaded to a web server.

The problem is when we execute the script as it is right now, when the runner is done, it will shut down, and we will lose the files generated by the job.

So let's modify this, on the Build steps let's modify the code.

```

build:
  needs: test
  runs-on: ubuntu-latest
  steps:
    - name: Get code
      uses: actions/checkout@v3

```

```
- name: Install dependencies
  run: npm ci
- name: Build website
  run: npm run build
- name: Upload Artifacts
  uses: actions/upload-artifact@v4
  with:
    name: dist-files
    path: dist
```

Explanation

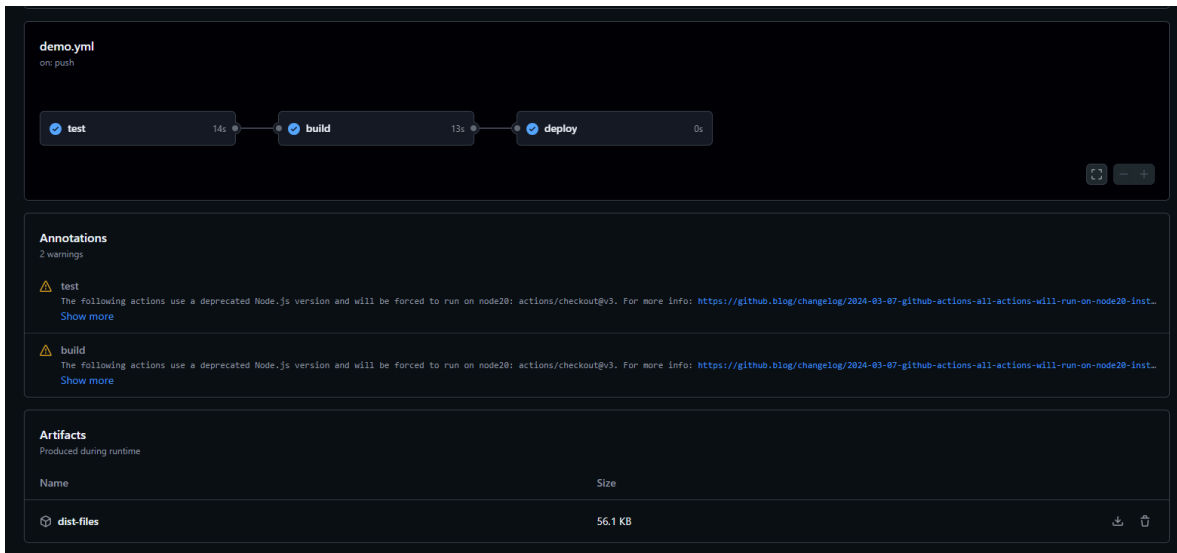
```
- name: Upload Artifacts
  uses: actions/upload-artifact@v4
```

We will give the step a name and specify that we are going to use an existing action which helps us upload the artifacts.

```
with:
  name: dist-files
  path: dist
```

we will give it a name, in this case its *dist-files*, and we will specify the path the files will be uploaded from *dist*.

and now in your workflow, you should be able to see and download the files:



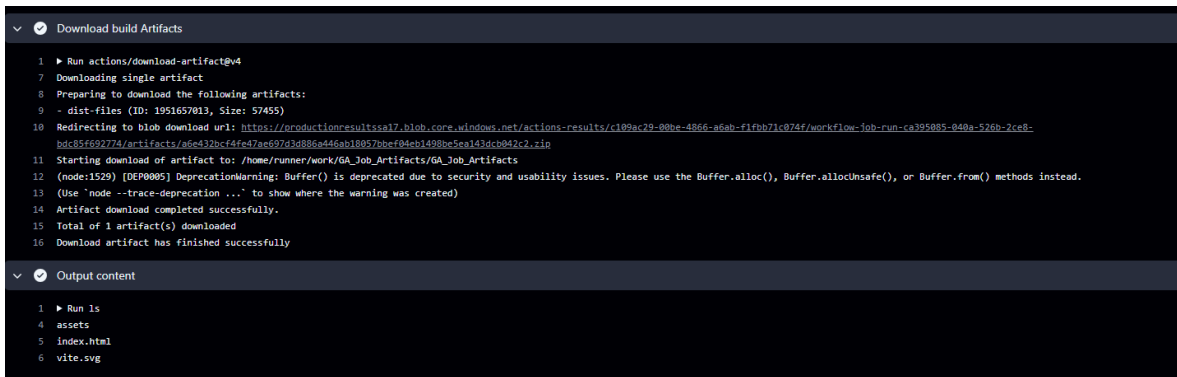
Artifacts

But we need to use the files in the deployment job. The files produced by the build process will not persist to the deployment job because they are on different runners even though they use the same runner definition. To get the artifacts, we need to download them in the deployment step. Let's modify the deployment job

```
deploy:
  needs: build
  runs-on: ubuntu-latest
  steps:
    - name: Download build Artifacts
      uses: actions/download-artifact@v4
      with:
        name: dist-files
    - name: Output content
      run: ls
    - name: Deploy
      run: echo "Deploying..."
```

We will add the download artifact action and specify the names of the files to be downloaded, in this case its *dist-files* which is the same name we used in the build step. The action here will download and unzip the files: And we can now use the 'ls' step to list the contents.

Now we can see the output right here:



```
Download build Artifacts

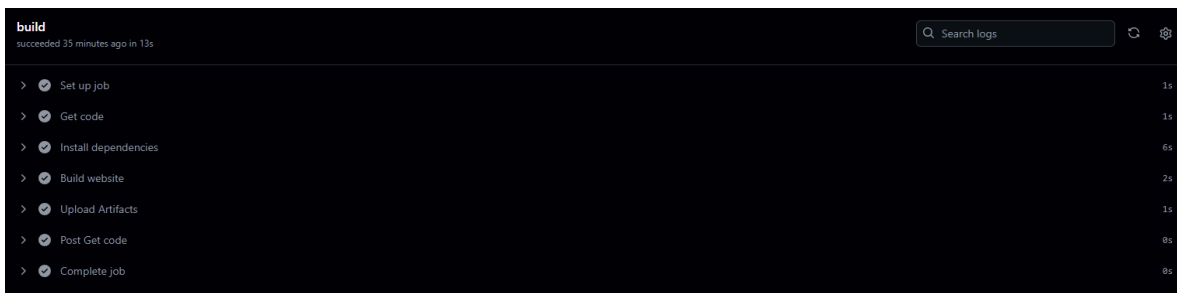
1 ▶ Run actions/download-artifact@v4
7 Downloading single artifact
8 Preparing to download the following artifacts:
9 - dist-files (ID: 1951657013, Size: 57455)
10 Redirecting to blob download url: https://productionresultsssl7.blob.core.windows.net/actions-results/c109ar29-00be-4866-a6ab-f1fb71c074f/workflow-job-run-ca395085-040a-526b-7ce8-bdc85f692774/artifacts/a6e432bcf4fe47ae69743d886e446ab18857bbef04eb1498be5e143dcb042c2.zip
11 Starting download of artifact to: /home/runner/work/GA_Job_Artifacts/GA_Job_Artifacts
12 (node:1529) [DEP0005] DeprecationWarning: Buffer() is deprecated due to security and usability issues. Please use the Buffer.alloc(), Buffer.allocUnsafe(), or Buffer.from() methods instead.
13 (Use 'node --trace-deprecation ...' to show where the warning was created)
14 Artifact download completed successfully.
15 Total of 1 artifact(s) downloaded
16 Download artifact has finished successfully

Output content

1 ▶ Run ls
4 assets
5 index.html
6 vite.svg
```

Download content

Dependencies Caching

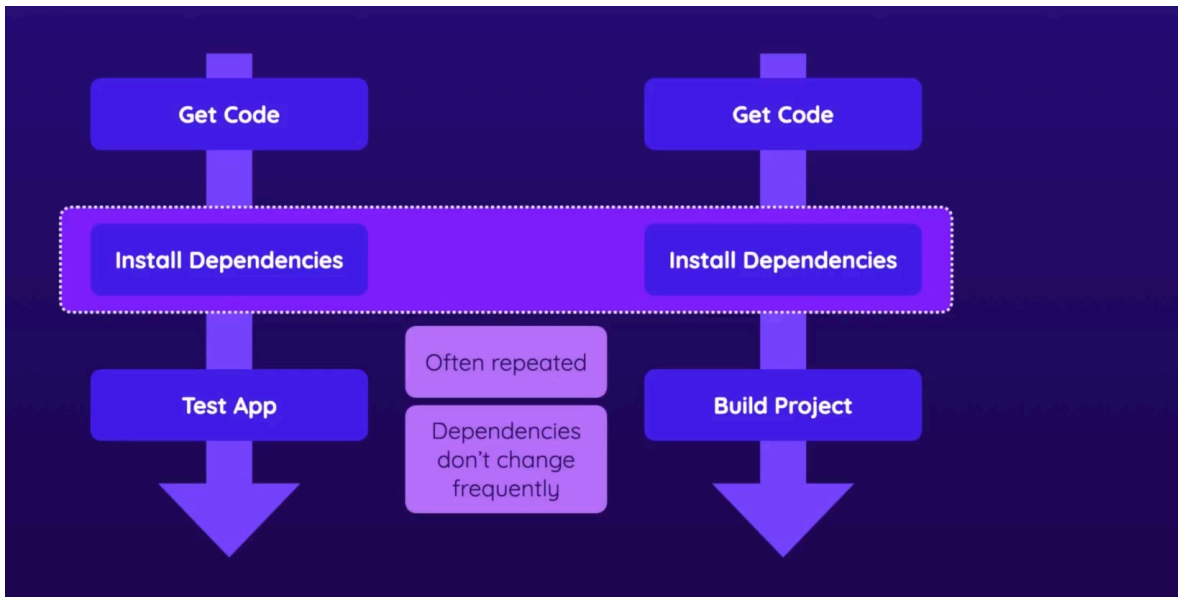


build	
succeeded 35 minutes ago in 13s	
> Set up job	1s
> Get code	1s
> Install dependencies	6s
> Build website	2s
> Upload Artifacts	1s
> Post Get code	0s
> Complete job	0s

Step Time

If we take a look at the build step, we will notice that the step that takes more time is the dependency installation step. Also, we will notice that we have repeated the step in both test and build job.

We can save some time if we cache the dependencies and reuse then in the build step.



Cache

GitHub Action has an action that will help us cache the dependencies

```
- name: Cache Dependencies
  uses: actions/cache@v4
  with:
    path: ~/.npm
    key: deps-node-modules- ${hashFiles('**/package-lock.json')}
```

This will now cache the dependencies and will only run *npm ci* if the *package-lock.json* file has changed so that means we will be able to use cache for every step provided the *package-lock.json* file has not changed.

Now the workflow file will look like:

```
jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - name: Get code
        uses: actions/checkout@v4
      - name: Cache Dependencies
        uses: actions/cache@v4
        with:
```

```

    path: ~/.npm
    key: deps-node-modules- ${{hashFiles('**/package-lock.json')}}
- name: Install dependencies
  run: npm ci
- name: Lint code
  run: npm run lint
- name: Test code
  run: npm run test

build:
  needs: test
  runs-on: ubuntu-latest
  steps:
    - name: Get code
      uses: actions/checkout@v4
    - name: Cache Dependencies
      uses: actions/cache@v4
      with:
        path: ~/.npm
        key: deps-node-modules- ${{hashFiles('**/package-lock.json')}}
    - name: Install dependencies
      run: npm ci
    - name: Build website
      run: npm run build
    - name: Upload Artifacts
      uses: actions/upload-artifact@v4
      with:
        name: dist-files
        path: dist

```

Now you will notice that the Cache is used for the second Job (build). And cache will be used, hence it will be a bit faster.

```
Cache Dependencies 1s
1 ▶ Run actions/cache@v4
9 Cache Size: ~19 MB (19568772 B)
10 /usr/bin/tar -xJf /home/runner/work/_temp/dfd364fa-ed51-4b9d-91d6-a2fd7f1172a/cache.tgz -P -C /home/runner/work/GA_Job_Artifacts/GA_Job_Artifacts --use-compress-program unxz
11 Cache restored successfully
12 Cache restored from key: deps-node-modules- 5a782fe93589e2cbe03aac182697ecd9156b0019da2b2cd4ec8f40591000c823

Install dependencies 4s
1 ▶ Run npm ci
4 added 375 packages, and audited 376 packages in 4s
5 77 packages are looking for funding
6   run 'npm fund' for details
7 12 vulnerabilities (5 moderate, 6 high, 1 critical)
8 To address all issues, run:
9   npm audit fix
10 Run 'npm audit' for details.

Build website 3s
```

Caching