Contents

[GITHUB ACTION 2](#_Toc188814545)

[KEY BUILDING BLOCKS 2](#_Toc188814546)

[WORKFLOW 2](#_Toc188814547)

[JOBS 2](#_Toc188814548)

[RUNNERS 3](#_Toc188814549)

[STEPS 3](#_Toc188814550)

[CREATING A WORKFLOW 4](#_Toc188814551)

[SIMPLE WORKFLOW 4](#_Toc188814552)

[RUNNING THE WORKFLOW 5](#_Toc188814553)

[EVENTS THAT TRIGGERED WORKFLOWS 6](#_Toc188814554)

[CREATING WORKFLOW FROM CODEBASE 7](#_Toc188814555)

[ACTIONS 7](#_Toc188814556)

[EXAMPLE 7](#_Toc188814557)

[MULTIPLE TRIGGERS 9](#_Toc188814558)

[RUNNING JOBS IN PARALLEL VS SEQUENTIAL 9](#_Toc188814559)

[RUNNING JOBS IN PARALLEL 9](#_Toc188814560)

[RUNNING JOBS SEQUENTIALLY 9](#_Toc188814561)

[EXPRESSIONS AND CONTEXT OBJECTS 10](#_Toc188814562)

[COMMONLY USED PROPERTIES OF THE CONTEXT OBJECT 11](#_Toc188814563)

[ACCESSING CONTEXT OBJECT USING EXPRESSION 11](#_Toc188814564)

[CONTEXT OBJECTS 11](#_Toc188814565)

[EXERCISE 11](#_Toc188814566)

[WORKFLOW TRIGGERS / EVENTS 14](#_Toc188814567)

[EVENT FILTERS AND ACTIVITY TYPES 15](#_Toc188814568)

[EVENT FILTERS 16](#_Toc188814569)

[ACTIVITY TYPES 17](#_Toc188814570)

[CANCELLING AND SKIPPING WORKFLOWS 20](#_Toc188814571)

[SKIPING WORKFLOW RUN 20](#_Toc188814572)

[JOB ARTIFACTS AND OUTPUTS 20](#_Toc188814573)

# GITHUB ACTION

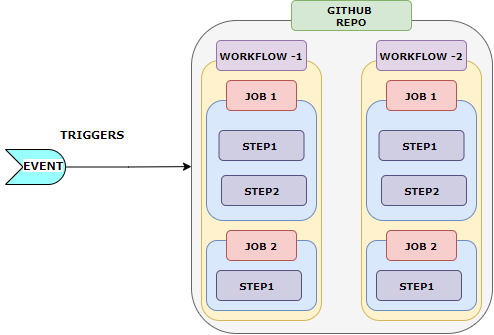
**GitHub Actions is a feature of the popular code hosting platform GitHub that allows users to automate various tasks and workflows directly within their repositories. It provides a way to build, test, and deploy code in a consistent and efficient manner**.

* With GitHub Actions, we can define custom workflows using YAML syntax, which are triggered by specific events such as pushes, pull requests, or scheduled intervals.
* These workflows can include a series of steps, which can be executed on different operating systems, environments, or even in parallel.

**COMMON USE CASES**

1. **Continuous Integration (CI):** Automatically building and testing code every time changes are made to a repository, helping to catch bugs and issues early on.
2. **Continuous Deployment (CD):** Deploying applications or services to various environments (such as staging or production) when specific conditions are met, such as passing CI tests or merging pull requests.
3. **Code quality and security checks**: Running linters, code formatters, vulnerability scanners, or other static analysis tools to ensure code quality and security standards are met.
4. **Release automation**: Automating the process of creating release notes, tagging versions, and publishing artifacts or packages.

## KEY BUILDING BLOCKS



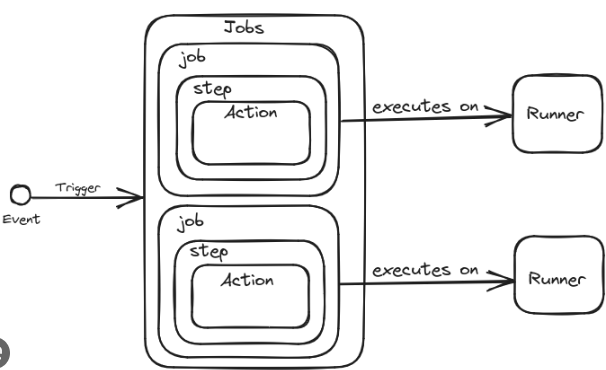
WORKFLOW

* A workflow is a configurable automated process that we can define in the repository using a YAML file**.** We can have multiple workflows defined in a repository.
* Workflows are attached to GitHub repositories, which contains one or more jobs.
* The workflow can be triggered by specific events, such as **push, pull request, or a scheduled interval**. These events are called **triggers**.

JOBS

* Within a workflow, we can define one or more jobs. Jobs contain the steps that will be executed.
* Based on configuration **- The Jobs can run in parallel(default), sequential or conditional**.
* Each job can have its own environment, such as the operating system, required tools, and dependencies.
* Every job defines a so-called runner which is simply the execution environment, the machine and operating system that will be used for executing these steps.

### RUNNERS



* A runner is a machine or virtual environment that executes a workflow. Runners are responsible for running the jobs defined in the workflow and executing the steps within those jobs.
* GitHub offers a selection of different runner types and operating systems, such as Ubuntu, Windows, and macOS.
* **Runners**: [actions/runner-images: GitHub Actions runner images](https://github.com/actions/runner-images)
* The **runs-on** keyword is used in a workflow file to specify the type of runner and operating system on which the job should run. It determines the environment in which the steps of a job will be executed.
* A job represents a set of steps that run on the same runner.

|  |  |
| --- | --- |
| **jobs:**  **my\_job:**  **runs-on: ubuntu-latest**  **steps:**  **- name: Step 1**  **run: echo "Hello, world!"**  **- name: Step 2**  **run: echo "This is another step"** | * In this example, the my\_job job will run on an Ubuntu runner. The runs-on keyword specifies the **ubuntu-latest** as runner, which is a GitHub-hosted runner with the latest version of Ubuntu. * By specifying the runs-on value, we can control the execution environment for our jobs and ensure that they run on the desired operating system and runner type. |

STEPS

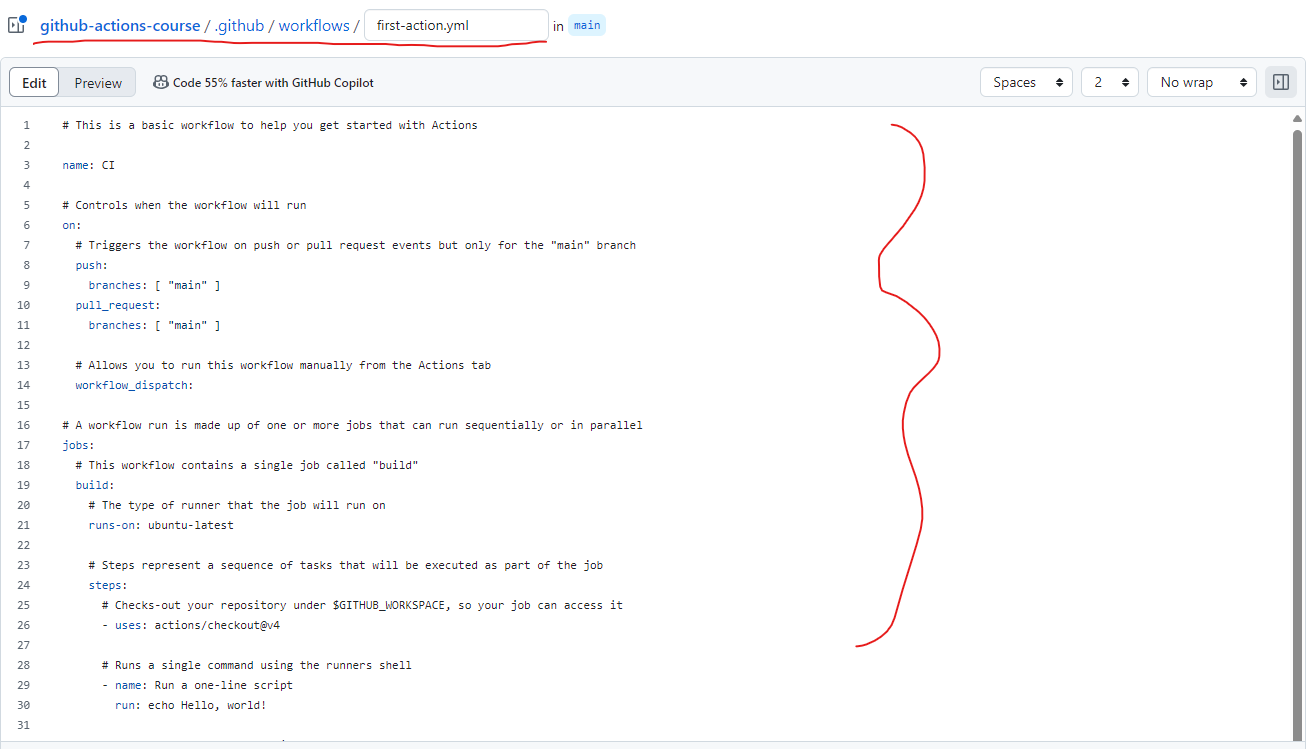
* Steps define the individual tasks that make up a job.
* **Steps can be a shell scripts or an Action** Each step runs in its own shell or container environment and can execute commands or actions.
* Steps can be written in various languages, such as JavaScript, Python, Ruby, or shell scripts.
* We can use pre-defined actions from the GitHub Marketplace or create our own custom actions to reuse across workflows.

## CREATING A WORKFLOW

|  |  |
| --- | --- |
|  | 1. Navigate to Repo 🡪 Actions 🡪 Select a Workflow template 2. This will create an editor with some initial workflow structure & content in “YAML” file |

1. The path where the work will be created is **<repo>/.github/workflows/<work-file-name>.yml**

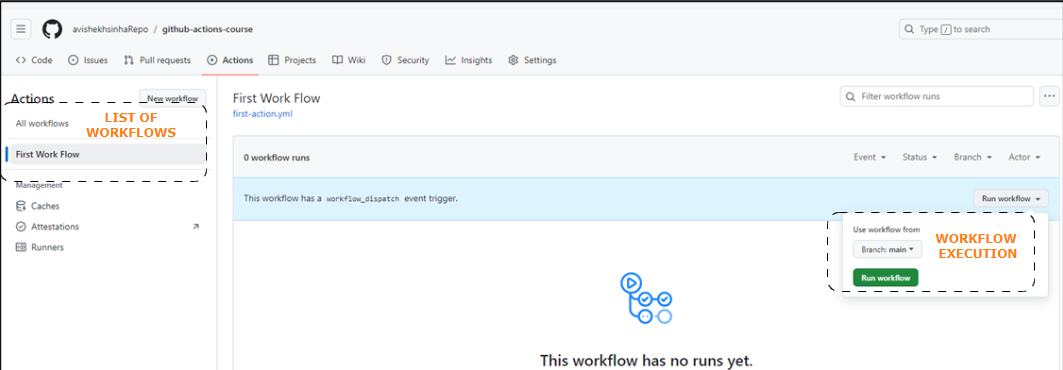
***Note****: We can create multiple workflows by creating multiple ‘yaml’ files in ‘workflow’ directory*

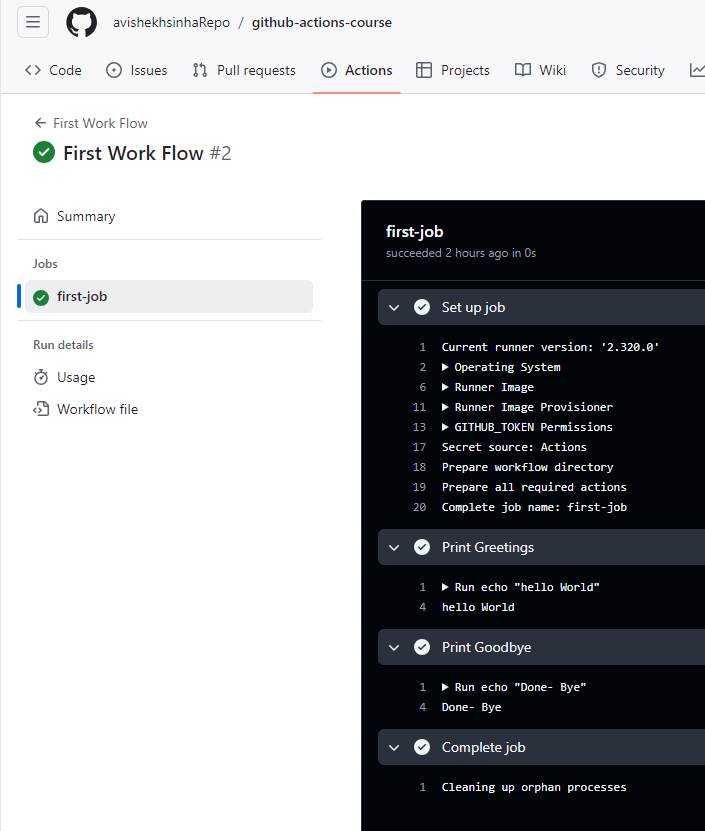


### SIMPLE WORKFLOW

|  |  |
| --- | --- |
| **name**: First Work Flow  **on**: workflow\_dispatch  **jobs**:  first-job:  **runs-on**: ubuntu-latest  **steps**:  - **name**: Print Greetings  **run**: echo "hello World"  - **name**: Print Goodbye  **run**: echo "Done- Bye"  *So, when we manually trigger the "First Work Flow" workflow, it will run the "first-job" job on the latest version of Ubuntu. The job consists of two steps: printing the greeting message "hello World" and printing the farewell message "Done- Bye" to the console.* | * **name**: First Work Flow: This sets the name of the workflow to "First Work Flow". * **on**:The "on" keyword in a GitHub Actions workflow configuration file specifies the event or set of events that will trigger the workflow. It determines when the workflow will be executed. * GitHub Actions supports various other events that can be used with the "on" keyword, such as:   + push: Triggered when code is pushed to the repository.   + pull\_request: Triggered when a pull request is created or updated.   + schedule: Triggered at specific scheduled intervals.   + repository\_dispatch: Triggered by a custom event using the GitHub API.   In the example   * **workflow\_dispatch**: This specifies that the workflow should be triggered manually using the "workflow\_dispatch" event. This means we can manually run this workflow from the GitHub Actions UI. * **jobs**: This section defines the jobs that will be executed as part of the workflow. In this case, there is one job named "first-job".   + **first-job**: This section defines the properties and steps for the "first-job" job.   + **runs-on**: ubuntu-latest:     - This specifies that the job should run on the latest version of the Ubuntu operating system.     - GitHub Actions provides a variety of runner environments to choose from, and in this case, it is using the Ubuntu environment. * **steps**: This section defines the individual steps that will be executed as part of the job.   + **name**: Print Greetings: This sets the name of the step to "Print Greetings".   + **run**: echo "hello World": This is the command that will be executed as part of the step. In this case, it is using the echo command to print the message "hello World" to the console.   + **name**: Print Goodbye: This sets the name of the step to "Print Goodbye".   + **run**: echo "Done- Bye": This is the command that will be executed as part of the step. It uses the echo command to print the message "Done- Bye" to the console. |

### RUNNING THE WORKFLOW





|  |
| --- |
| NOTE  If we need to run multiple shell commands (or multi-line commands, e.g., for readability), we can easily do so by adding the pipe symbol (|) as a value after the run: key.  Like this:   1. ... 2. run: | 3. echo "First output" 4. echo "Second output"   **This will run both commands in one step.** |

## EVENTS THAT TRIGGERED WORKFLOWS

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

## CREATING WORKFLOW FROM CODEBASE

|  |  |
| --- | --- |
|  | To organize the work flow with the Repo   * We can create a **.github** folder at the root level of the repository. * Inside the .github folder, we can create a workflows folder to store all workflow files. * Each workflow file should have a “yml” extension and contain the definition of a specific workflow. * We can create multiple workflow files within the workflows folder to define different workflows for different purposes; |

## ACTIONS

* In GitHub Actions, actions are the individual units of work/application that performs a (typically complex) frequently repeated task ,that can be combined to create workflows.
* Actions are reusable, standalone tasks that can be executed within a workflow.
* **They are defined in YAML files and can be created by the GitHub community or by the repository owner.**
* Actions can perform a wide range of tasks, **such as building and testing code, deploying applications, sending notifications, or running custom scripts**. They are designed to automate various parts of the software development lifecycle.
* Actions are typically written in JavaScript, but they can also be created in other languages like Python or Shell. **They can be triggered by events, such as a push to a repository, a pull request, or a scheduled interval.**
* When defining a workflow in GitHub Actions, we can use pre-built actions from the GitHub Marketplace (<https://github.com/marketplace?type=actions> ) or create our own custom actions. These actions can be combined and configured to create a workflow that fits your specific needs.
* Actions in GitHub Actions provide a way to modularize and reuse common tasks, making workflows more efficient and easier to maintain. They contribute to the automation and streamlining of various development processes, enabling teams to achieve continuous integration and delivery.

### EXAMPLE

In this example – we want to run the test cases using Github Action for a React Application

#### STEPS IN GITHUB ACTION

The workflow run in runner machine/ environment hence we need following steps in the workflow

|  |  |
| --- | --- |
|  | 1. **Download /checkout the code in Runner environment** 2. **Install the required node version** 3. **Install the dependencies** 4. **Run the test cases** |

|  |  |
| --- | --- |
| **name: The Test Case Workflow**  **on: [push]**  **jobs:**  **test:**  **runs-on: ubuntu-latest**  **steps:**  **- name: Get Code**  **uses: actions/checkout@v3**  **- name: Install Node JS**  **uses: actions/setup-node@v2**  **with:**  **node-version: '18'**  **- name: Install Dependencies**  **run: npm ci**  **- name: Run Tests**  **run: npm test** | * The workflow is triggered by a push event to the repository, meaning it will run whenever changes are pushed to the repository. * The workflow consists of a single job named test, which runs on the latest version of Ubuntu (ubuntu-latest). The job is broken down into several steps: * **Get Code**:   + This step uses the **actions/checkout@v3** action to check out the repository's code.   + This action ensures that the workflow has access to the codebase that triggered the workflow. * **Install Node JS**:   + This step uses the actions/setup-node@v2 action to set up Node.js version 18.   + This action configures (using **with**) the environment with the specified version of Node.js, which is necessary for running npm commands and the project's JavaScript code. * **Install Dependencies**:   + This step runs the **npm ci** command to install the project's dependencies.   + The npm ci command is optimized for continuous integration environments and ensures that the exact versions of dependencies specified in the [package-lock.json](vscode-file://vscode-app/c:/Program%20Files/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-sandbox/workbench/workbench.html) file are installed. * **Run Tests**   + The final step runs the npm test command to execute the project's test suite. This command typically runs any tests defined in the project, ensuring that the code changes do not introduce any regressions or issues. |

**NOTE**

|  |  |
| --- | --- |
| on:  push:  branches:  **- 'main'**  **- 'releases/\*\*'** | * **This trigger will run the workflow when the code is pushed to main or to a branch that starts with releases/.** |

**npm ci**

* npm ci is a command that is used to install dependencies for a Node.js project based on the package-lock.json.
* It is primarily used in continuous integration (CI) or automated build environments to ensure reproducible and deterministic dependency installations.

**How npm ci works?**

* It deletes the node\_modules directory and any existing package-lock.json or npm-shrinkwrap.json file.
* It installs the exact versions of dependencies specified in the package-lock.json or npm-shrinkwrap.json file, ensuring that the installed dependencies match the exact versions specified in the lock file. This helps to create a consistent and reproducible environment for your project.
* It does not modify the package.json file or update the versions of the dependencies listed in it. Instead, it strictly relies on the lock file to install the dependencies.
* Unlike npm install, which allows for more flexibility in updating dependencies based on the semver ranges specified in the package.json file, npm ci is designed to be used in CI or automated build environments where reproducibility and consistency are essential.
* It is recommended to run npm ci instead of npm install in your CI pipeline or when setting up a new development environment to ensure that the exact versions of dependencies are installed as specified in the lock file.

## MULTIPLE TRIGGERS

|  |  |
| --- | --- |
| **on: [push,workflow\_dispatch]** | * We are allowing the workflow to be triggered both by a push event on the specified branch and through manual triggering via the workflow dispatch event. |

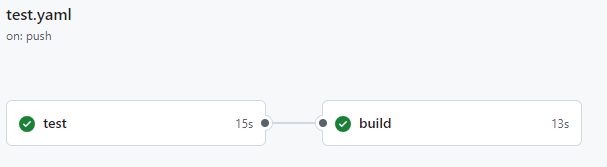
## RUNNING JOBS IN PARALLEL VS SEQUENTIAL

### RUNNING JOBS IN PARALLEL

|  |  |
| --- | --- |
|  | * Parallel execution allows multiple jobs to run simultaneously, potentially reducing the overall execution time of your workflow. * This is useful when the jobs are independent of each other and can be executed concurrently without any dependencies or conflicts. For example - tasks like building and testing multiple configurations or running tests in parallel against different environments. |

### RUNNING JOBS SEQUENTIALLY

* Sequential execution ensures that jobs are executed one after another, in a defined order.
* This is useful when the jobs have dependencies on each other, and the output of one job is required as an input for the next job.
* Sequential jobs can be defined by specifying the **needs** property in the workflow file, which ensures that a job runs only after its dependencies have completed successfully.
* Sequential jobs can be helpful when you need to perform tasks like deploying an application after successful build and test jobs, or running a set of steps in a specific order.



|  |  |
| --- | --- |
|  | `name: The Test Case and Deployment Workflow  on: [push,workflow\_dispatch]  jobs:  test:  runs-on: ubuntu-latest  steps:  - name: Get Code  uses: actions/checkout@v3  - name: Install Node JS  uses: actions/setup-node@v4  with:  node-version: '18'  - name: Install Dependencies  run: npm ci  - name: Run Tests  run: npm test  build:  **needs: test**  runs-on: ubuntu-latest  steps:  - name: Get Code  uses: actions/checkout@v3  - name: Install Node JS  uses: actions/setup-node@v4  with:  node-version: '18'  - name: Install Dependencies  run: npm ci  - name: Build Project  run: npm run build |
| Note:  We can use the needs keyword in GitHub Actions to wait for multiple jobs to finish before starting a subsequent job or step.  **needs: [job1, job2]** |

## EXPRESSIONS AND CONTEXT OBJECTS

* The context object in GitHub Actions provides information about the execution environment and the event that triggered the workflow.
* It contains several properties that you can access within the workflow to retrieve details about the repository, the event, and other relevant information.

### COMMONLY USED PROPERTIES OF THE CONTEXT OBJECT

|  |  |
| --- | --- |
| **github.repository** | Contains information about the repository where the workflow is running, such as the repository owner and name. |
| **github.event** | Represents the payload of the GitHub event that triggered the workflow run. This includes data specific to the event type, such as push, pull request, issue, etc |
| **github.sha** | Specifies the commit SHA associated with the event that triggered the workflow run. |
| **github.ref** | Provides the Git reference (branch or tag) associated with the event that triggered the workflow run. |
| **github.actor** | Represents the username of the user or the name of the GitHub App that initiated the event |

### ACCESSING CONTEXT OBJECT USING EXPRESSION

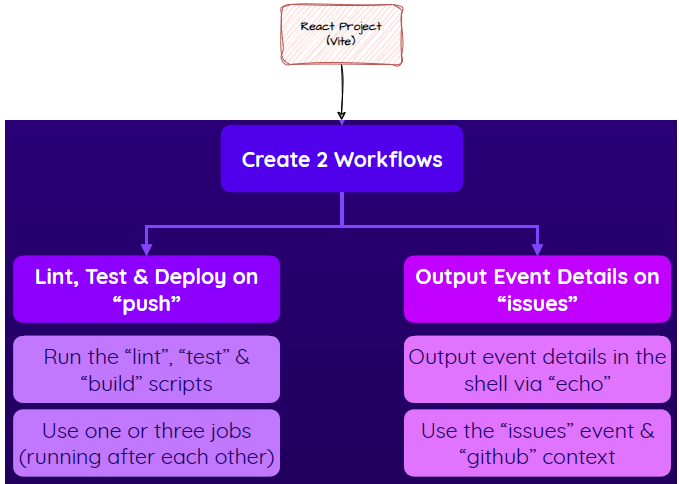
* We can access these properties using the github object in the workflow steps.
* **For example, github.repository would be accessed as ${{ github.repository }}.**

### CONTEXT OBJECTS

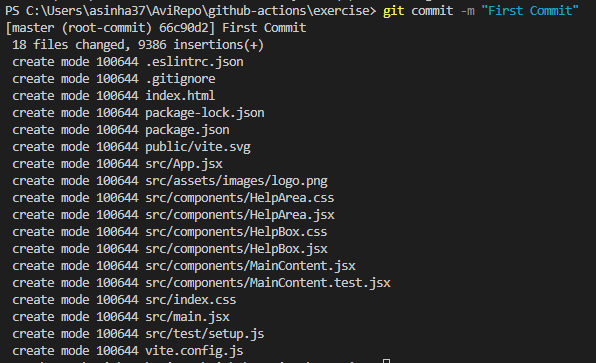
**REFERENCE**

* **Context Object** : <https://docs.github.com/en/enterprise-cloud@latest/actions/writing-workflows/choosing-what-your-workflow-does/accessing-contextual-information-about-workflow-runs#about-contexts>
* **Expression**: <https://docs.github.com/en/enterprise-cloud@latest/actions/writing-workflows/choosing-what-your-workflow-does/evaluate-expressions-in-workflows-and-actions#about-expressions>

## EXERCISE



1. Step 1: **CREATE A NEW REMOTE REPO**. Example: [**github-action-react**](https://github.com/avishekhsinhaRepo/github-action-react) - <https://github.com/avishekhsinhaRepo/github-action-react.git>
2. Step 2: Initialize the local repo : **git init**
3. **Step 3: Commit the code** 
   1. **git add .**
   2. **git commit -m “First Commit”**



1. Step 4: Push to the remote repository
   1. git remote add origin <https://github.com/avishekhsinhaRepo/github-action-react.git>
   2. git branch -M main
   3. git push -u origin main
2. Step 5: Create a workflow using “Simple Workflow” template

A screenshot of a computer

Description automatically generated

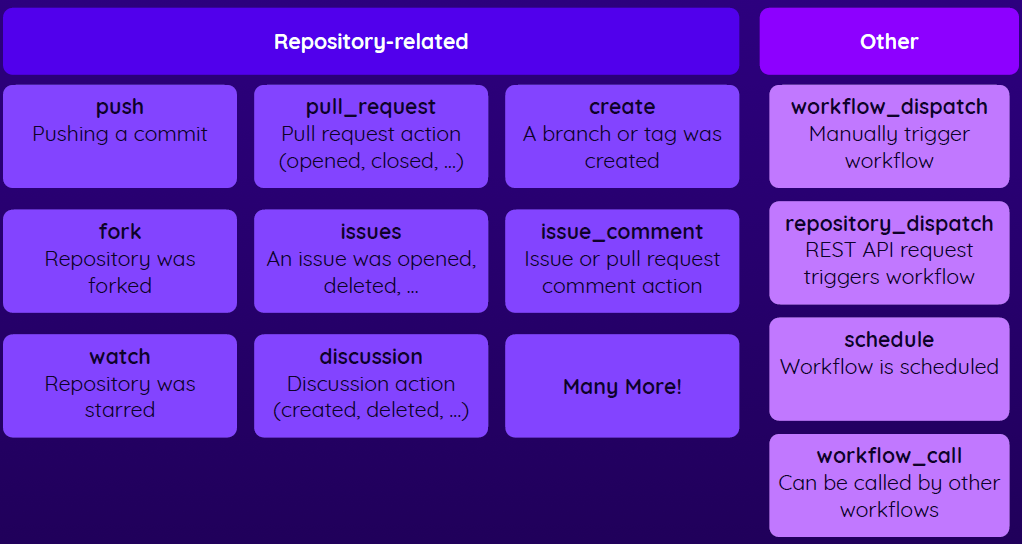
|  |  |
| --- | --- |
| **Release.yaml** | Package.json |
| **name: React App Test Build and deploy Workflow**  **on: [workflow\_dispatch, push]**  **jobs:**  **lint:**  **runs-on: ubuntu-latest**  **steps:**  **- name: Checkout code**  **uses: actions/checkout@v2**  **- name: Install dependencies**  **run: npm ci**  **- name: Run tests**  **run: npm run lint**  **test:**  **runs-on: ubuntu-latest**  **steps:**  **- name: Checkout code**  **uses: actions/checkout@v2**  **- name: Install dependencies**  **run: npm ci**  **- name: Run tests**  **run: npm test**  **build:**  **needs: test**  **runs-on: ubuntu-latest**  **steps:**  **- name: Checkout code**  **uses: actions/checkout@v2**  **- name: Install dependencies**  **run: npm ci**  **- name: Build**  **run: npm run build**  **deploy:**  **needs: build**  **runs-on: ubuntu-latest**  **steps:**  **- name: Deploy**  **run: echo "Deployed"** | **{**  **"name": "02-basic-example",**  **"private": true,**  **"version": "0.0.0",**  **"type": "module",**  **"scripts": {**  **"dev": "vite",**  **"lint": "eslint --ext .jsx --fix src",**  **"build": "vite build",**  **"preview": "vite preview",**  **"test": "vitest run"**  **},**  **"dependencies": {**  **"prop-types": "^15.8.1",**  **"react": "^18.2.0",**  **"react-dom": "^18.2.0"**  **},**  **"devDependencies": {**  **"@testing-library/jest-dom": "^5.16.5",**  **"@testing-library/react": "^13.3.0",**  **"@testing-library/user-event": "^14.4.3",**  **"@types/react": "^18.0.17",**  **"@types/react-dom": "^18.0.6",**  **"@vitejs/plugin-react": "^2.0.1",**  **"eslint": "^8.23.0",**  **"eslint-plugin-react": "^7.31.1",**  **"jsdom": "^20.0.0",**  **"vite": "^3.0.7",**  **"vite-plugin-eslint": "^1.8.1",**  **"vitest": "^0.22.1"**  **}**  **}** |
| **ISSUE WORKFLOW**  **name: React issues Workflow**  **on:**  **issues:**  **types:**  **- opened**  **- closed**  **jobs:**  **react-issues:**  **runs-on: ubuntu-latest**  **steps:**  **- name: Checkout code**  **uses: actions/checkout@v2**  **- name: Install dependencies**  **run: npm ci**  **- name: Run tests**  **run: npm run test** | * It will be triggred whenever a New Issue is opened or existing issue is closed. |

A screenshot of a computer

Description automatically generated

# WORKFLOW TRIGGERS / EVENTS

* Events are triggers that initiate the execution of workflows.
* The events are mostly either **repository related** like push, pull\_request or **other events like workflow\_dispatch**
* Documentation : <https://docs.github.com/en/actions/writing-workflows/choosing-when-your-workflow-runs/events-that-trigger-workflows>



|  |  |
| --- | --- |
| MULTPLE TRIGGERS FOR A WORKFLOW | name: Events Demo 1  on**: [workflow\_dispatch, push]**  jobs:  …. |

## EVENT FILTERS AND ACTIVITY TYPES

**A screenshot of a computer

Description automatically generated**

* **EVENT FILTERS AND ACTIVITY TYPES ARE USED TO DEFINE SPECIFIC CONDITIONS THAT MUST BE MET FOR A WORKFLOW TO RUN**
* **Events can have activity types and filters.**
* Event filters and activity types are defined in the "on" section of the workflow file, where you specify the event that triggers the workflow

### EVENT FILTERS

* **Event filters allow us to specify conditions related to the event that triggers a workflow**.
* **Filter Pattern CheatSheet** : <https://docs.github.com/en/actions/writing-workflows/workflow-syntax-for-github-actions#filter-pattern-cheat-sheet>

#### COMMONLY USED EVENT FILTERS

|  |  |
| --- | --- |
| **BRANCHES**   * We can specify one or more branches using glob patterns or regular expressions. * Workflows will only run if the event occurs on the specified branch(es). * **- main**: This line specifies that the workflow should trigger when a push event occurs on the "main" branch. * **- 'feature/\*'**: This line specifies that the **workflow should trigger when a push event occurs on any branch that starts with "feature/".** * **- 'release/\*\*'**: This line specifies that the workflow should trigger when a **push event occurs on any branch that starts with "release/" and has any number of subdirectories.** * **- 'dev-\*'**: This line specifies that the workflow should trigger when a push event occurs on any branch that starts with "dev-". | **on:**  **push:**  **branches:**  **- main**  **- ‘feature/\*’**  **- ‘release/\*\*’**  **- ‘dev-\*’**   * Workflow will run when a push event occurs on the "**main**" branch or **any branch starting with "feature/".** |
| **Note**   * The characters \*, [, and ! are special characters in YAML. If we start a pattern with \*, [, or !, we must enclose the pattern in quotes. * Also, if we use a flow sequence with a pattern containing [ and/or ], the pattern must be enclosed in quotes | **# Valid**  **paths:**  **- '\*\*/README.md'**  **# Invalid - creates a parse error that**  **# prevents your workflow from running.**  **paths:**  **- \*\*/README.md**  **# Valid**  **branches: [ main, 'release/v[0-9].[0-9]' ]**  **# Invalid - creates a parse error**  **branches: [ main, release/v[0-9].[0-9] ]** |
| **TAGS**   * We can specify one or more tags using glob patterns or regular expressions. * Workflows will only run if the event occurs on a tag that matches the specified pattern(s). | **on:**  **push:**  **tags:**  **- v1.\***  **- v2.0**   * The workflow will run when a push event occurs on a tag that matches the pattern "v1.\*" or the exact tag "v2.0". |
| **PATHS**   * We can specify one or more file paths using glob patterns. * Workflows will only run if the event involves changes to files that match the specified path(s). | **on:**  **push:**  **paths:**  **- 'src/\*\*'**   * The workflow will run when a push event involves changes to files in the "src" directory or any of its subdirectories. |

### ACTIVITY TYPES

* **ACTIVITY TYPES ALLOW US TO FURTHER FILTER WORKFLOWS BASED ON THE SPECIFIC ACTIVITY THAT TRIGGERED THE EVENT.**
* Here are some commonly used activity types:

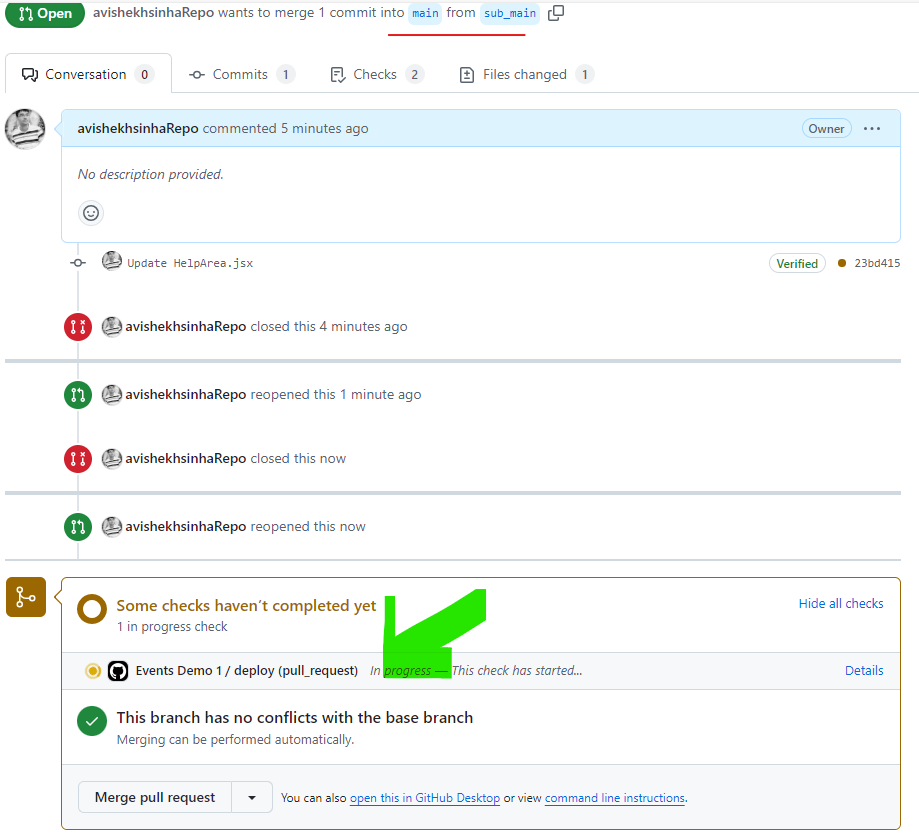
Created: This activity type filters workflows based on whether a new branch, tag, or pull request was created.

Pushed: This activity type filters workflows based on whether a commit was pushed to the repository.

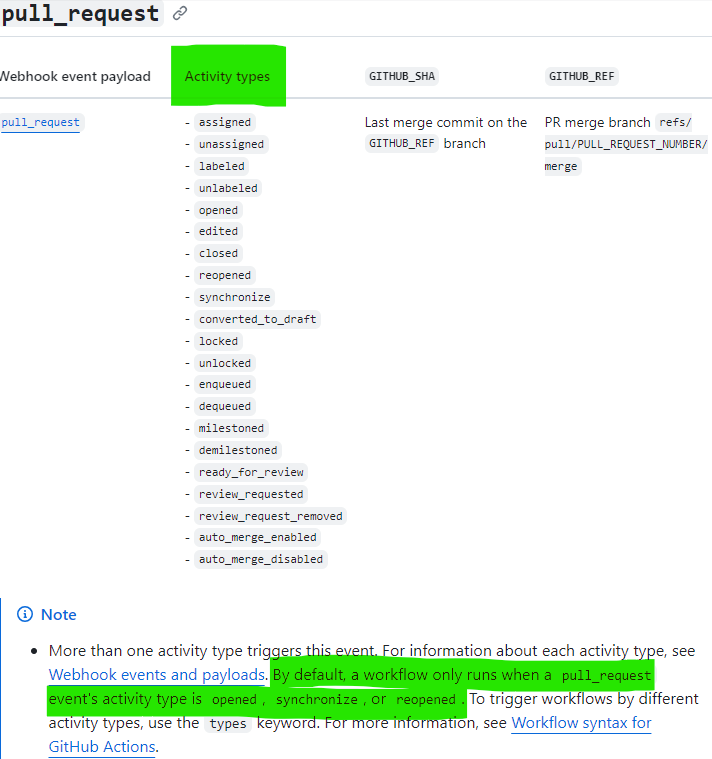
Closed: This activity type filters workflows based on whether an issue, pull request, or check run was closed.

Opened: This activity type filters workflows based on whether an issue, pull request, or check run was opened.

|  |  |  |
| --- | --- | --- |
| **VARIATION 1** | **VARIATION 2** | **VARIATION 3** |
| **name: Events Demo 1**  **on:**  **pull\_request:**  **types:**  **- opened**  **- synchronize**  **- reopened**  **…** | **name: Events Demo 1**  **on:**  **pull\_request:**  **types: [opened, synchronize, reopened]**  **…** | **name: Events Demo 1**  **on:**  **pull\_request:**  **types: opened**  **..** |



SAMPLE ACTIVITY TYPE DOCUMENTATION



|  |
| --- |
| * By combining event filters and activity types, we can create more fine-grained rules for when workflows should run. For example, we can configure a workflow to only run when a commit is pushed to the "master" branch and the changes involve files in a specific directory. |

EXAMPLE

|  |
| --- |
| name: Events Demo 1  on:    pull\_request:      types:        - opened        - synchronize        - reopened      branches:        - main        - release/\*\*    push:      branches:        - main        - release/\*\*    workflow\_dispatch: |

EXAMPLE -2



## CANCELLING AND SKIPPING WORKFLOWS

|  |  |
| --- | --- |
|  | * By default, the workflow gets cancelled of Job fails * By default, a Job fails if at least one step fails * MANUALALLY CANCELLING WORKFLOW   Screenshot showing the summary for a workflow that is currently running. The "Cancel workflow" button is highlighted with a dark orange outline. |

### SKIPING WORKFLOW RUN

* Documentation : <https://docs.github.com/en/actions/managing-workflow-runs-and-deployments/managing-workflow-runs/skipping-workflow-runs>
* We can add a special comment in our commit message to skip a workflow run. if we add any of the following strings to the commit message in a push the workflow run will be skipped.

1. [skip ci]
2. [ci skip]
3. [no ci]
4. [skip actions]
5. [actions skip]

|  |  |
| --- | --- |
| EXAMPLE | git commit -m "event added **[skip ci**]" |

# JOB ARTIFACTS(DATA) AND OUTPUTS

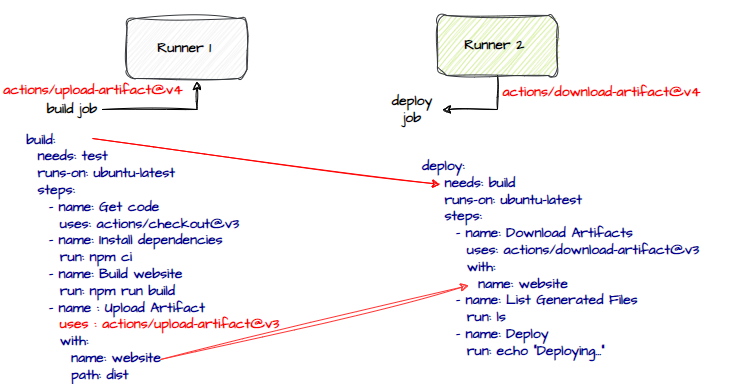
## JOB ARTIFACTS

* **Job artifacts are files or directories generated during the execution of a job**
* We can use the Job artifacts to persist and make available for download or use in subsequent steps or workflows.

A diagram of a job

Description automatically generated

### USE CASE/ EXAMPLE



* When the jobs runs, they run on their own runner environments
* Now – let’s consider an application (e.g a React Application) which has **build** job and a **deploy** job .
* The use case, where job artifact can be leveraged can be **the artifacts which get generated in build steps(compiled JS and HTML) can be used in the deploy step** so that it can be deployed to a web server

|  |  |
| --- | --- |
| **name: Deploy website**  **on:**  **push:**  **branches:**  **- main**  **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**  **- name : Upload Artifact**  **uses : actions/upload-artifact@v3**  **with:**  **name: website**  **path: dist**  **deploy:**  **needs: build**  **runs-on: ubuntu-latest**  **steps:**  **- name: Download Artifacts**  **uses: actions/download-artifact@v3**  **with:**  **name: website**  **- name: List Generated Files**  **run: ls**  **- name: Deploy**  **run: echo "Deploying..."** | **build**   * **Step: Upload Artifact**:   + This step uses the `**actions/upload-artifact**` action to upload the `dist` directory (built website) as an artifact named "website".     **deploy**   * `**Download Artifacts**`: This step uses the `**actions/download-artifact**` action to download the artifact named "website" that was uploaded in the `build` job. * `List Generated Files`: This step runs the `ls` command to list the files in the downloaded artifact directory. * The `build` job builds the website, uploads the `dist` directory as an artifact, and depends on the successful completion of the `test` job. * The `deploy` job downloads the artifact from the `build` job, lists the generated files, and can be extended to include deployment logic. **The `deploy` job depends on the successful completion of the `build` job.** |

A screenshot of a computer

Description automatically generated

## JOB OUTPUTS

* We can use job outputs to pass data or values between jobs within the same workflow.
* Job outputs allow us to share information generated in one job with subsequent jobs.