Continuous integration (CI) is a software practice that requires frequently committing code to a shared repository. Committing code more often detects errors sooner and reduces the amount of code a developer needs to debug when finding the source of an error. Frequent code updates also make it easier to merge changes from different members of a software development team. This is great for developers, who can spend more time writing code and less time debugging errors or resolving merge conflicts.

When you commit code to your repository, you can continuously build and test the code to make sure that the commit doesn't introduce errors. Your tests can include code linters (which check style formatting), security checks, code coverage, functional tests, and other custom checks.

Building and testing your code requires a server. You can build and test updates locally before pushing code to a repository, or you can use a CI server that checks for new code commits in a repository.

You can configure your CI workflow to run when a GitHub event occurs (for example, when new code is pushed to your repository), on a set schedule, or when an external event occurs using the repository dispatch webhook.

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. When a test fails, one of your changes may have caused the failure.

When you set up CI in your repository, GitHub analyzes the code in your repository and recommends CI workflows based on the language and framework in your repository.

### Enabling notifications

If you enable email or web notifications for GitHub Actions, you'll receive a notification when any workflow runs that you've triggered have completed. The notification will include the workflow run's status (including successful, failed, neutral, and canceled runs). You can also choose to receive a notification only when a workflow run has failed.

When ever we do any thing on git hub webhooks are fired and which is basically a request which we can send to the server listening it .Normally server infrastructure cost are high including a hardware where the system needs to installed , static connections and security. So git actions is the solutgion for it which basically helps listen to those web hooks from git and impletment what ever action you want it to perform from git to git.You can integrate your custom apps with github by listening to events. Where you can run your commands to what ever machine windows,mac or linux you want which are basically running as virtual machines at back end in github.From repository automation to CI/CD implementation ,build and test functionality every thing is present in git action and one click away.

### Automation of our workflows and postproduction testing and deployment are just one click away. You just have to make workflows which are basically store in .github/workflows/(nameofworkflow).yml like for example , I have setup up a workflow in my git repo for a basic react app <https://github.com/muhammadumerhack/CI-CD-tutorial> the workflows are stored in .github/workflows/main.yml main.yml file is where it gets stored

### Name:

The name of your workflow. GitHub displays the names of your workflows on your repository's actions page. If you omit name, GitHub sets it to the workflow file path relative to the root of the repository.

**On:**

**Required** The name of the GitHub event that triggers the workflow. You can provide a single event string, array of events, array of event types, or an event configuration map that schedules a workflow or restricts the execution of a workflow to specific files, tags, or branch changes. For a list of available events, see "[Events that trigger workflows](https://docs.github.com/en/articles/events-that-trigger-workflows).

Single event

# Trigger on push

on: push

List of events

# Trigger the workflow on push or pull request

on: [push, pull\_request]

Its is a trigger on which the workflow is set to start like pull ,push

Its is defined like

On:

Push:

Braches:[master]  
branches tells which that if the push event occurs at that branch so start the workflow

Jobs

A workflow run is made up of one or more jobs. Jobs run in parallel by default. To run jobs sequentially, you can define dependencies on other jobs using the jobs.<job\_id>.needs keyword.

Each job runs in an environment specified by runs-on.

You can run an unlimited number of jobs as long as you are within the workflow usage limits.

### Usage Limits

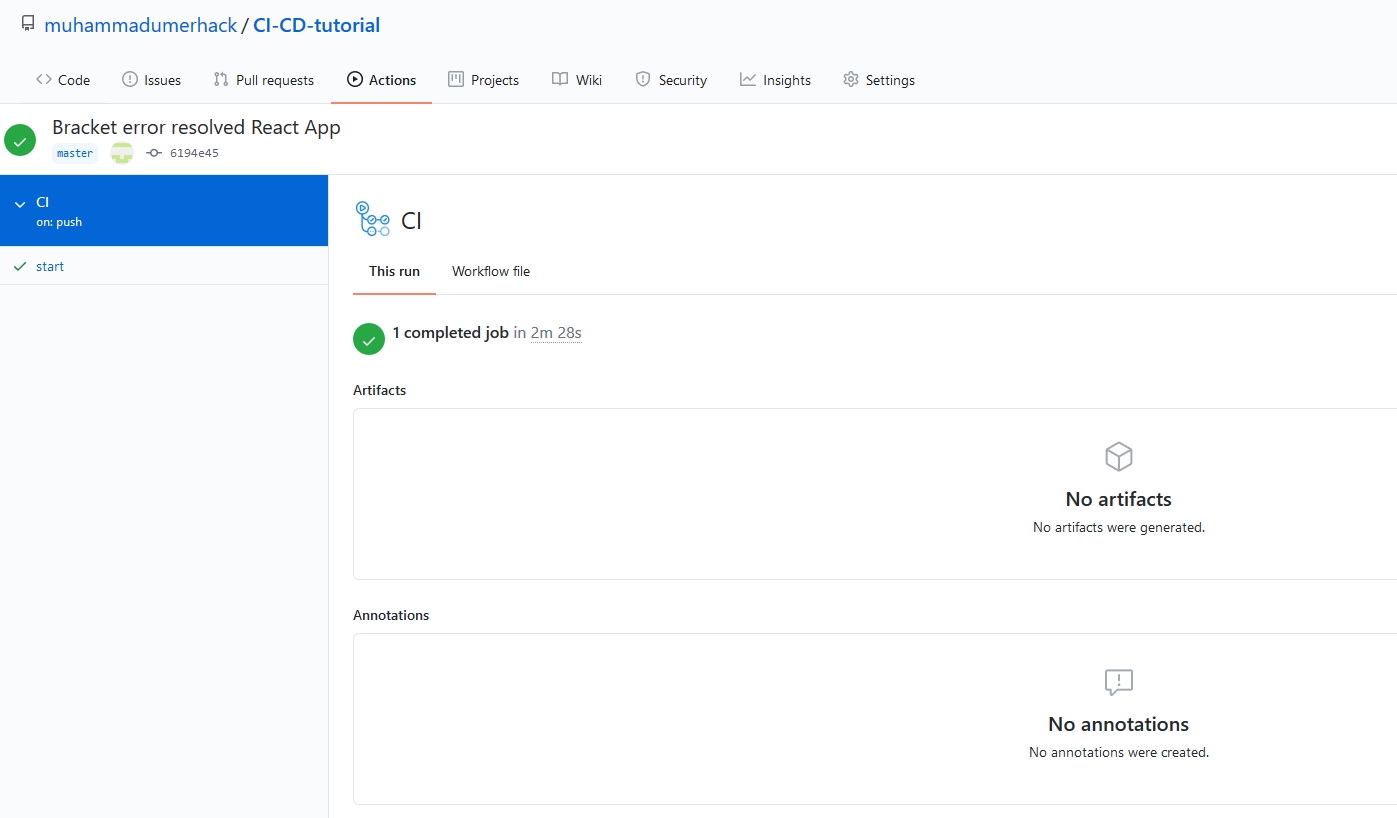
There are some limits on GitHub Actions usage, and will vary depending on whether you use GitHub-hosted or self-hosted runners. These limits are subject to change.

* **Job execution time** - Each job in a workflow can run for up to 6 hours of execution time. If a job reaches this limit, the job is terminated and fails to complete. This limit does not apply to self-hosted runners.
* **Workflow run time** - Each workflow run is limited to 72 hours. If a workflow run reaches this limit, the workflow run is cancelled. This limit also applies to self-hosted runners.
* **Job queue time** - Each job for self-hosted runners can be queued for a maximum of 24 hours. If a self-hosted runner does not start executing the job within this limit, the job is terminated and fails to complete. This limit does not apply to GitHub-hosted runners.
* **API requests** - You can execute up to 1000 API requests in an hour across all actions within a repository. If exceeded, additional API calls will fail, which might cause jobs to fail. This limit also applies to self-hosted runners.
* **Concurrent jobs** - The number of concurrent jobs you can run in your account depends on your GitHub plan, as indicated in the following table. If exceeded, any additional jobs are queued. There are no concurrency limits for self-hosted runners.

|  |  |  |
| --- | --- | --- |
| GitHub plan | Total concurrent jobs | Maximum concurrent macOS jobs |
| Free | 20 | 5 |
| Pro | 40 | 5 |
| Team | 60 | 5 |
| Enterprise | 180 | 50 |

* **Job matrix** - A job matrix can generate a maximum of 256 jobs per workflow run. This limit also applies to self-hosted runners.

### Click here to check out list of [Webhook events](https://docs.github.com/en/actions/reference/events-that-trigger-workflows#webhook-events)



### ENV

A map of environment variables that are available to all jobs and steps in the workflow. You can also set environment variables that are only available to a job or step. For more information, see [jobs.<job\_id>.env](https://docs.github.com/en/actions/reference/workflow-syntax-for-github-actions#jobsjob_idenv) and [jobs.<job\_id>.steps.env](https://docs.github.com/en/actions/reference/workflow-syntax-for-github-actions#jobsjob_idstepsenv).

When more than one environment variable is defined with the same name, GitHub uses the most specific environment variable. For example, an environment variable defined in a step will override job and workflow variables with the same name, while the step executes. A variable defined for a job will override a workflow variable with the same name, while the job executes.

#### [Example](https://docs.github.com/en/actions/reference/workflow-syntax-for-github-actions#example)

env:

SERVER: production

### [jobs.<job\_id>](https://docs.github.com/en/actions/reference/workflow-syntax-for-github-actions#jobsjob_id)

Each job must have an id to associate with the job. The key job\_id is a string and its value is a map of the job's configuration data. You must replace <job\_id> with a string that is unique to the jobs object. The <job\_id> must start with a letter or \_ and contain only alphanumeric characters, -, or \_.

#### [Example](https://docs.github.com/en/actions/reference/workflow-syntax-for-github-actions#example-2)

jobs:

my\_first\_job:

name: My first job

my\_second\_job:

name: My second job

### [jobs.<job\_id>.name](https://docs.github.com/en/actions/reference/workflow-syntax-for-github-actions#jobsjob_idname)

The name of the job displayed on GitHub.

### [jobs.<job\_id>.needs](https://docs.github.com/en/actions/reference/workflow-syntax-for-github-actions#jobsjob_idneeds)

Identifies any jobs that must complete successfully before this job will run. It can be a string or array of strings. If a job fails, all jobs that need it are skipped unless the jobs use a conditional statement that causes the job to continue.

#### [Example](https://docs.github.com/en/actions/reference/workflow-syntax-for-github-actions#example-3)

jobs:

job1:

job2:

needs: job1

job3:

needs: [job1, job2]

In this example, job1 must complete successfully before job2 begins, and job3 waits for both job1 and job2 to complete.

The jobs in this example run sequentially:

1. job1
2. job2
3. job3

### jobs.<jobid>.needs

**Required** The type of machine to run the job on. The machine can be either a GitHub-hosted runner, or a self-hosted runner.

Git hub self hosted runners

If you use a GitHub-hosted runner, each job runs in a fresh instance of a virtual environment specified by runs-on.

Available GitHub-hosted runner types are:

| **Virtual environment** | **YAML workflow label** |
| --- | --- |
| Windows Server 2019 | windows-latest or windows-2019 |
| Ubuntu 20.04 | ubuntu-20.04 |
| Ubuntu 18.04 | ubuntu-latest or ubuntu-18.04 |
| Ubuntu 16.04 | ubuntu-16.04 |
| macOS Catalina 10.15 | macos-latest or macos-10.15 |
| Example: runs-on: ubuntu-latest |  |
|  |  |

### Steps

A job contains a sequence of tasks called steps. Steps can run commands, run setup tasks, or run an action in your repository, a public repository, or an action published in a Docker registry. Not all steps run actions, but all actions run as a step. Each step runs in its own process in the runner environment and has access to the workspace and filesystem. Because steps run in their own process, changes to environment variables are not preserved between steps. GitHub provides built-in steps to set up and complete a job.

You can run an unlimited number of steps as long as you are within the workflow usage limits.

Example:

name: Greeting from Mona

on: push

jobs:

my-job:

name: My Job

runs-on: ubuntu-latest

steps:

- name: Print a greeting

env:

MY\_VAR: Hi there! My name is

FIRST\_NAME: Mona

MIDDLE\_NAME: The

LAST\_NAME: Octocat

run: |

echo $MY\_VAR $FIRST\_NAME $MIDDLE\_NAME $LAST\_NAME.

Conditional Steps:

You can use the if conditional to prevent a step from running unless a condition is met. You can use any supported context and expression to create a conditional.

steps:

- name: My first step

if: ${{ github.event\_name == 'pull\_request' && github.event.action == 'unassigned' }}

run: echo This event is a pull request that had an assignee removed.