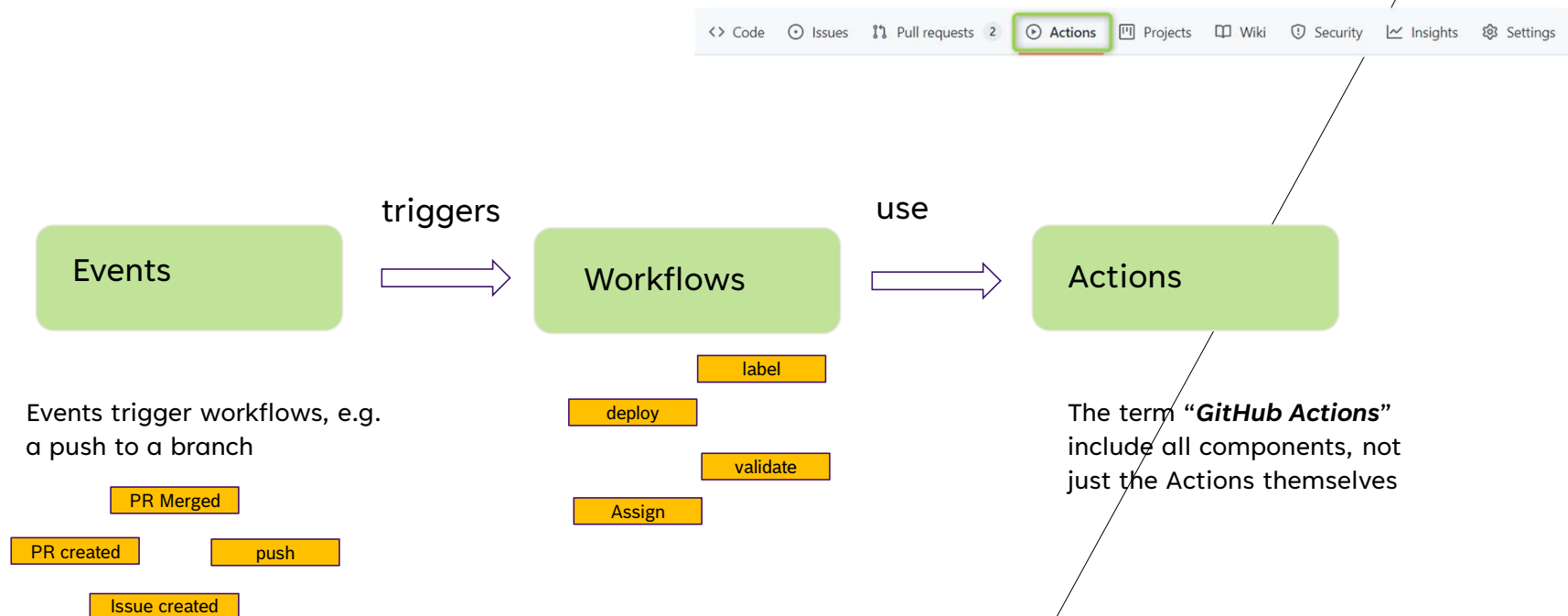


# GitHub Actions

Keerthi V

# WHAT IS GITHUB ACTIONS?

1. Actions help you automate tasks within your software development
2. They are event driven.



# Advantages

Integration with GitHub

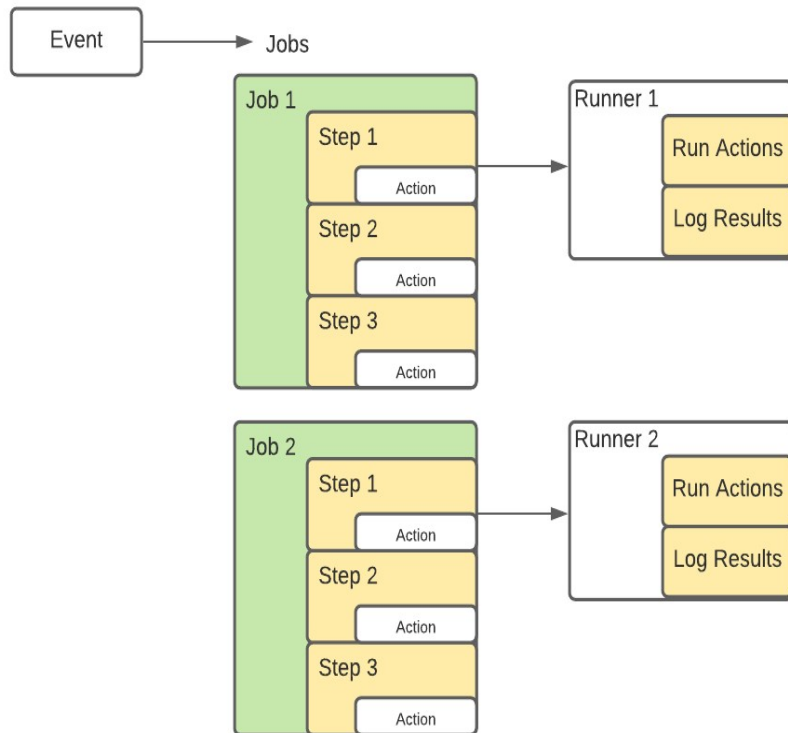
Automate everything within the GitHub flow

GitHub flow with templates built by developers (Ready actions in Marketplace)

We can also create your own custom CI workflows

virtual machines on multiple operating system

## Key Components



<repo>.github/workflows/main.yml

```
3  name: CI
4
5  # Controls when the workflow will run
6  on:
7    # Triggers the workflow on push or pull request events but only for the "main" branch
8    push:
9      branches: [ "main" ]
10   pull_request:
11     branches: [ "main" ]
12
13   # Allows you to run this workflow manually from the Actions tab
14   workflow_dispatch:
15
16   # A workflow run is made up of one or more jobs that can run sequentially or in parallel
17   jobs:
18     # This workflow contains a single job called "build"
19     build:
20       # The type of runner that the job will run on
21       runs-on: [ linux ]
22
23       # Steps represent a sequence of tasks that will be executed as part of the job
24       steps:
25         # Checks-out your repository under $GITHUB_WORKSPACE, so your job can access it
26         - uses: actions/checkout@v3
27
28         # Runs a single command using the runners shell
29         - name: Run a one-line script
30           run: echo Hello, world!
31
32         # Runs a set of commands using the runners shell
33         - name: Run a multi-line script
34           run: |
35             echo Add other actions to build,
36             echo test, and deploy your project.
```

# Events

An Event is specific activity that triggers a workflow.

```
.github > workflows > ! main.yml
1  name: CI
2
3  on:
4    push:
5      branches: [ "main" ]
6    pull_request:
7      branches: [ "main" ]
8    schedule:
9      - cron: 0 12 * * 1
10
11  workflow_dispatch:
12
13  jobs:
14    build:
15      runs-on: [ linux ]
16      steps:
17        # $GITHUB_WORKSPACE
18        - uses: actions/checkout@v3
19
20
21        - name: Run a one-line script
22          run: echo Hello, world!
23
24        - name: Run a multi-line script
25          run: |
26            echo Add other actions to build,
27            echo test, and deploy your project.
```

# Events

schedule

```
name: Do things every 5 minutes
on:
  schedule:
    - cron: "* /5 * * * *
```

pull\_request

```
on:
  pull_request:
    types: [opened, synchronize, edited, reopened]
```

push

```
on:
  pull_request:
    branches-ignore:
      - 'development'
```

```
on:
  push:
```

```
on:
  push:
    branches:
      - main
      - 'releases/**'
```

# Events

**workflow\_dispatch**

**repository\_dispatch**

**workflow\_call**

on:  
  workflow\_dispatch:

on:  
  repository\_dispatch:  
    types: [test\_result]

on: workflow\_call

# Context

Contexts : Contexts are a way to access information about workflow runs, variables, runner env, jobs, and steps.

- GitHub
- Runner
- ENV
- Secrets
- Needs



## GitHub Context :

When you run any workflow, Github Actions exposes a set of default environment variables.

```
1 ▶ Run echo "HOME: ${HOME}"
13 HOME: /home/Azurelogin
14 GITHUB_WORKFLOW: Environment variables
15 GITHUB_REPOSITORY: /githubactions
16 GITHUB_EVENT_NAME: workflow_dispatch
17 GITHUB_WORKSPACE: /home/Azurelogin/_work/githubactions/githubactions
18 GITHUB_SHA: b33fe249b487e0a51522db02099fbe86cee70783
19 GITHUB_REF: refs/heads/main
20 GITHUB_ACTION: __run
21 GITHUB_ACTIONS: true
```

```
jobs:
  context:
    runs-on: linux
  steps:
    - name: Run normal CI
      run: |
        echo github.workspace: ${github.workspace}
        echo github.ref_name: ${github.ref_name}
        echo github.event_name: ${github.event_name}
        echo github.actor: ${github.actor}

    - run: echo 'Hi ${env.name}' # Hi keerthi
    - run: echo 'Hi ${env.name}' # Hi kim
  env:
    name: kim
```

# ENV

## SCOPE :

- The entire **workflow**, by using env at the top level of the workflow file.
- The contents of a **job** within a workflow, by using jobs.<job\_id>.env.
- A specific **step** within a job, by using jobs.<job\_id>.steps[\*].env.

```
on:
  workflow_dispatch:

env:
  name: keerthi

jobs:
  context:
  runs-on: linux
  steps:
    - name: Run normal CI
      run: |
        echo github.workspace: ${github.workspace}
        echo github.ref_name: ${github.ref_name}
        echo github.event_name: ${github.event_name}
        echo github.actor: ${github.actor}

    - run: echo 'Hi ${env.name}' # Hi keerthi
    - run: echo 'Hi ${env.name}' # Hi kim
      env:
        name: kim

envcontext:
  runs-on: linux
  env:
    name: ki
  steps:
    - run: echo 'Hi ${env.name}' ki
```

# secrets

GitHub Secrets are encrypted and allow you to store sensitive information, such as access tokens, in your repository.

## steps:

- name: Azure details  
with: # Set Azure credentials secret as an input  
credentials: `${{ secrets.AZURE_CREDENTIALS }}`
- env: # Or as an environment variable  
credentials: `${{ secrets.AZURE_CREDENTIALS }}`

The screenshot shows the GitHub repository settings page for 'Actions secrets'. The left sidebar contains a navigation menu with options: General, Access, Collaborators, Code and automation, Branches, Tags, Actions, Webhooks, Pages, Security, Code security and analysis, Deploy keys, Secrets, Actions, and Dependabot. The 'Actions secrets' section is active, displaying a message: 'There are no secrets for this repository. Encrypted secrets allow you to store sensitive information, such as access tokens, in your repository.' A 'New repository secret' button is located in the top right corner. Below the message, the 'Actions secrets / New secret' form is visible, featuring a 'Name' field with the value 'AZURE\_CREDENTIALS' and a 'Secret' field. A green 'Add secret' button is positioned at the bottom of the form.

# Expressions

if

```
steps:  
  - name: Configuration for main branch  
    if: ${{ github.ref == 'refs/heads/main' }}
```

needs

```
jobs:  
  job1:  
  job2:  
    needs: job1  
  job3:  
    needs: [job1, job2]
```

Continue-on-error: true

```
- name: Job fail  
  continue-on-error: true  
  run |  
    exit 1  
- name: Next job  
  run |  
    echo Hello
```

# Status check Functions

success()

returns true if no previous steps have failed or have been canceled.

always()

returns true no matter if a step failed or the workflow was cancelled.

failure()

returns true if any previous step failed.

```
jobs:
  build:
    runs-on: [ linux ]
    steps:
      - name: Run a one-line script
        # run: exit 1
        run: echo Hello World

      - name: always exp
        if: ${{ always() }}
        #if: success() || failure()
        run: |
          echo will run always

      - name: The job has succeeded
        if: ${{ success() }}
        run: echo JOB has succeeded

      - name: The job has failed
        if: ${{ failure() }}
        run: echo JOB has failed
```

# Environments

<> Code Issues Pull requests Actions Projects Wiki Security Insights **Settings**

General Environments New environment

You can configure environments with protection rules and secrets. [Learn more.](#)

PROD	1 protection rule	1 secret	
DEV		1 secret	

Access

Collaborators

Code and automation

Branches

Tags

Actions

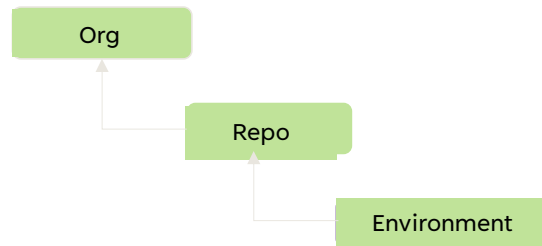
Hooks

**Environments**

Pages

Security

Code security and analysis



```
1 name: environment and Protection
2
3 on:
4   workflow_dispatch:
5
6
7 jobs:
8   greet:
9     runs-on: [ windows ]
10    outputs:
11      environment: ${{ steps.check.outputs.BRANCH_ENV }}
12    steps:
13      - name: Check environment
14        id: check
15        run: |
16          if ($env:GITHUB_REF -ne "refs/heads/main") {
17            echo "IS NOT main branch"
18            echo "::set-output name=BRANCH_ENV::DEV"
19          }
20          else {
21            echo "IS main branch"
22            echo "::set-output name=BRANCH_ENV::PROD"
23          }
24        shell: powershell
25    check:
26      runs-on: [ Windows ]
27      needs: greet
28      environment: ${{ needs.greet.outputs.environment }}
29      steps:
30        - name: env names
31          run: |
32            echo ${{ needs.greet.outputs.environment }}
33            echo "$SECRETS_CONTEXT"
34          env:
35            SECRETS_CONTEXT: ${{ toJson(secrets) }}
```

# Matrix

A matrix strategy lets you use variables in a single job definition to automatically create multiple job runs that are based on the combinations of the variables.

For example, you can use a matrix strategy to test your code in multiple versions of a language or on **multiple operating systems**

```
jobs:
  build:
    runs-on: ${ matrix.os }
    strategy:
      matrix:
        os: [linux, windows]
```

The screenshot displays the GitHub Actions interface for a workflow run. At the top, it shows 'Matrix Matrix #1' with a green checkmark. Below this, a 'Summary' tab is active, showing a list of jobs: 'build (linux)' and 'build (windows)', both with green checkmarks. To the right, a table provides details: 'Manually triggered 16 seconds ago', 'Status: Success', 'Total duration: 14s', and 'Artifacts: -'. The main section shows the workflow file '07\_matrix.yml' with the trigger 'on: workflow\_dispatch'. A 'Matrix: build' section lists the matrix jobs: 'build (linux)' (3s) and 'build (windows)' (8s), both with green checkmarks. The interface is clean and modern, with a light blue and white color scheme.

Matrix Matrix #1

Summary

Jobs

- build (linux)
- build (windows)

Manually triggered 16 seconds ago

Status: Success

Total duration: 14s

Artifacts: -

07\_matrix.yml  
on: workflow\_dispatch

Matrix: build

- build (linux) 3s
- build (windows) 8s

# Runner

## GitHub-hosted runners

To use a GitHub-hosted runner, create a job and use runs-on to specify the type of runner that will process the job, such as ubuntu-latest, windows-latest, or macos-latest.



## Self-hosted runners

You can host your own runners and customize the environment used to run jobs in your GitHub Actions workflow

- Repository-level runners are dedicated to a single repository. (You must be the repository owner)
- Organization-level runners can process jobs for multiple repositories in an organization. ( You must be the organization owner)
- Enterprise-level runners can be assigned to multiple organizations in an enterprise account. (organization owner, or have admin access to the repository.)





# Marketplace Actions

- uses: actions/checkout@v3
- uses: actions/javascript-action@v1.0.1

```
jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      # This step checks out a copy of your repository.
      - uses: actions/checkout@v3
      # This step references the directory that contains the action.
      - uses: ../github/actions/hello-world-action
```

<https://github.com/marketplace>

## Reusable workflow features

A reusable workflow is similar like any GitHub Actions workflow .

One difference is it includes a “**workflow\_call**” trigger.

```
on:  
  workflow_call:
```

You can then reference this workflow in another workflow

```
Uses:  
USER_OR_ORG_NAME/REPO_NAME/.github/workflows/REUSABLE_WORKFLOW_FILE.yml@TAG_OR_BRANCH
```

No workflow Duplication

Well tested

centrally maintained

DRY

Abstract geometric lines in white on a black background, forming various polygons and intersecting lines.

THANK YOU

**Workflow :**

- Workflows are basically pipelines or automated procedure.
- They are made up of one or more jobs and can be scheduled or triggered by an event
- Workflows are defined in the .github/workflows directory

**Jobs :**

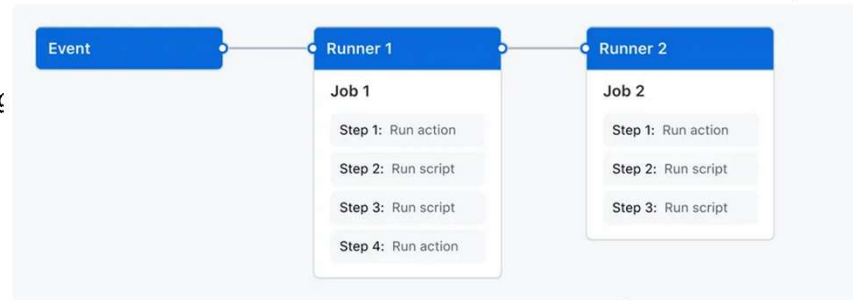
- A job is a set of steps that execute on the same runner.

**Steps:**

- A step is an individual task that can run commands in a job
- A step can be either an action or a shell command

**Runner :**

- Runners are processes on a server that run the workflow when it's triggered.
- You can use a runner hosted by GitHub or Self hosted.



## GITHUB\_TOKEN secret

GitHub automatically creates a unique **GITHUB\_TOKEN** secret to use in your workflow.

Examples :

- include passing the token as an input to an action
- using it to make an authenticated GitHub API request

# Matrix

A matrix strategy lets you use variables in a single job definition to automatically create multiple job runs that are based on the combinations of the variables.

For example, you can use a matrix strategy to test your code in multiple versions of a language or on **multiple operating systems**

## Using a single-dimension matrix

```
jobs:
  build:
    runs-on: ${{ matrix.os }}
    strategy:
      matrix:
        os: [linux, windows]
```

## Using a multi-dimension matrix

```
jobs:
  example_matrix:
    strategy:
      matrix:
        os: [ubuntu-22.04, ubuntu-20.04]
        version: [10, 12, 14]
    runs-on: ${{ matrix.os }}
    steps:
      - uses: actions/setup-node@v3
        with:
          node-version: ${{ matrix.version }}
```