GitHub Actions - Basic

Osama Tahir

Software Engineer at ASML

Introduction – About me

Engineer

Mentor

Traveller

Gamer

Artist

Researcher

Agenda

- What is GitHub Action
- Yaml... What is that?
- General Terms
- Types of Actions
- Types of Runners
- Flow of GitHub Actions
- Structure of Actions
- Conditions
- Cron Job
- Workflow Dispatch
- Composite Workflows – Structure
- Reusable Workflows – Structure

- Before Lunch

- Best Practices
- GitHub Secrets – Practical Example
- Practical Examples
- Competition

- After Lunch

What is GitHub Action?

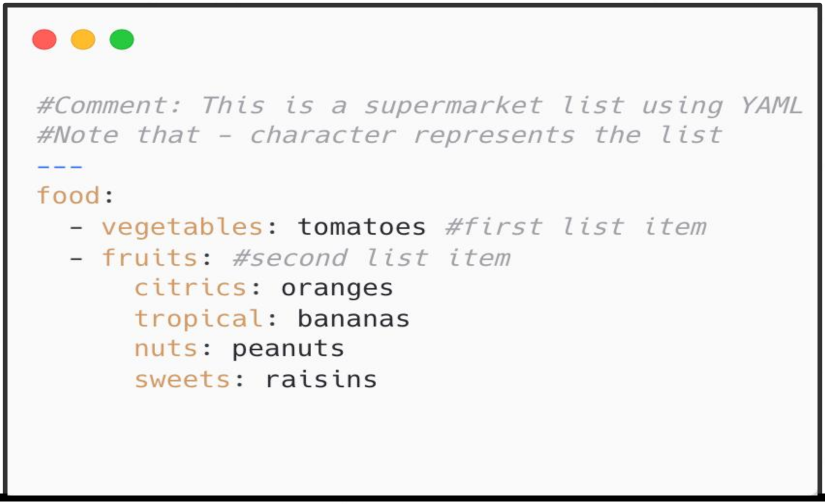
GitHub Actions is a continuous integration and continuous delivery platform that allows you to automate your build, test, and deployment pipeline. Main components includes:

- **Workflows**
- **Events**
- **Jobs**
- **Actions**
- **Runners**

Uses **YAML** file to define configuration....but wait, what is **YAML**?

YAML (Yet Another Markup Language)…What is that?

- **YAML Structure:** Uses indentation to represent hierarchy
- **Map:** Contain maps (key-value pairs) and lists (sequences of items)
- **Map Nesting:** It must be unique, and order does not matter
- **Mapping:** New map is started by increasing indentation or closing the previous map
- **Lists:** Lists use dashes and represent ordered sequences



```yaml
#Comment: This is a supermarket list using YAML
#Note that - character represents the list
---
food:
  - vegetables: tomatoes #first list item
  - fruits: #second list item
      citrics: oranges
      tropical: bananas
      nuts: peanuts
      sweets: raisins
```

To learn more about YAML

General Terms

Term	Definition
GitHub Actions	The name for the CI/CD system built into GitHub
Event	A trigger (like a push or pull request) that starts a workflow
Job	The set of steps to provide to a runner
Step	The fundamental units of work in a job
Runner	The cloud* resources used to run the steps of a job
Action	A reusable unit of code that performs a specific task in a workflow
Reusable workflow	A YAML file defining automation made up of one or more jobs triggered by events
Workflow	Either an entry workflow or a reusable workflow
Composite action	A series of actions bundled together as a single step
Workflow template	A reusable starting point for creating standard workflows across repositories

Types of Runners

There are **two type of runners** in GitHub Actions:

GitHub-hosted runners are managed by GitHub. It comes pre-installed with popular tools such as Python and NodeJs. Some examples include ubuntu-latest, windows-latest, macos-latest.

Self-hosted runners can be set up on your own machines, on-premise or cloud. These runners provide custom environments, more control, or saving cost on compute time.

Types of Actions

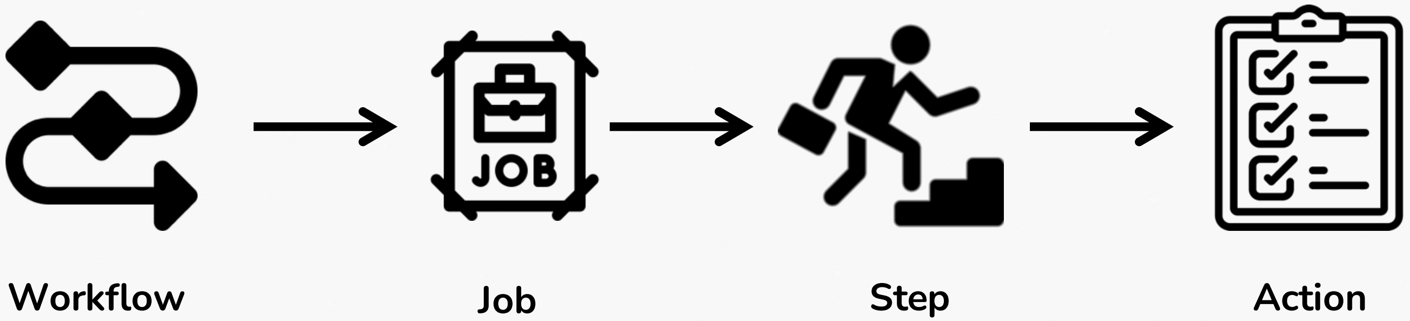
There are **three main types of actions** in GitHub Actions:

Docker Container Actions: These actions runs in a Docker container with a full environment. These are great for custom dependencies or runtime requirements (example in practical)

JavaScript Actions: These actions runs directly in the GitHub-hosted runner environment. These are fast and portable, and ideal for quick tasks (example in practical)

Composite Actions: These actions combines multiple shell script steps into one reusable action. These are useful for sharing sets of commands or workflows. (example in practical)

Flow of GitHub Actions



- An event triggers the workflow which contains the job. The job uses the steps to dictate which action will run
- A workflow requires at least requires one job and there can be multiple event triggers (example in practical)
- A job requires a runner to “run-on”
- ***But how a workflow looks like? Let's find out in next slide.......***

Structure of Workflow

Event: Trigger on every push to the main branch

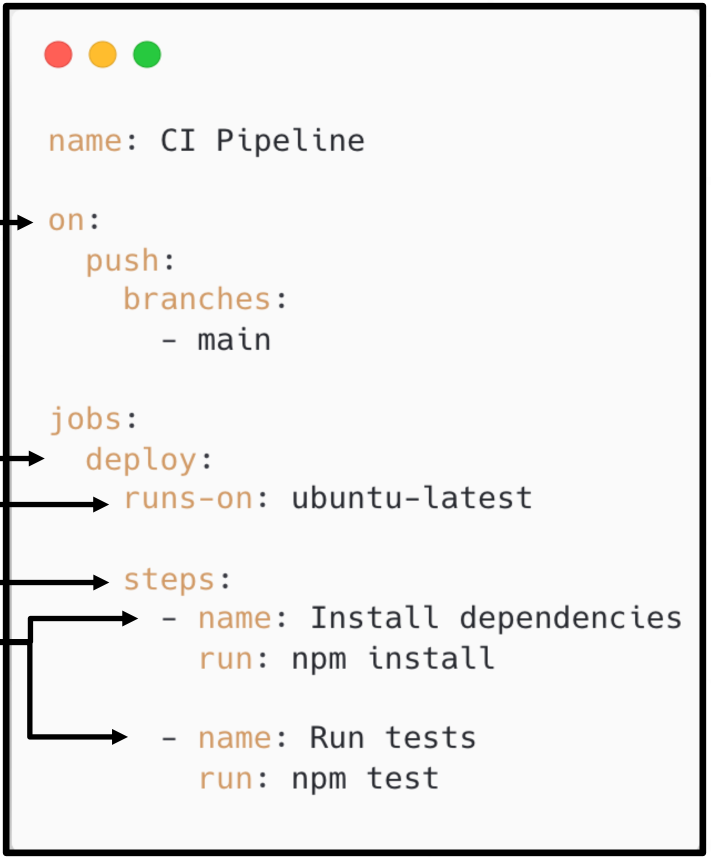
Job: The name of the job is “deploy”

Runner: GitHub-hosted runner, “ubuntu-latest”

Step: Steps in the job

Actions: ”Install dependencies” and “Run tests”

.github/workflows/ci.yaml



```yaml
name: CI Pipeline

on:
  push:
    branches:
      - main

jobs:
  deploy:
    runs-on: ubuntu-latest

    steps:
      - name: Install dependencies
        run: npm install

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

Conditions

It prevent a job from running unless conditions are met and <job_id> is used for conditions to prevent a job from running unless a condition is met. It supports any context and expression to create a condition.

```
name: my workflow
on: push
jobs:
  if: github.ref_name == 'main'
  test:
    runs-on: ubuntu-latest
    steps:
      - name: Execute tests
        run: exit 0
```

← **Condition** with context information

*To know more about GitHub **context information***

Cron Job

A **cron job** on GitHub Actions is a way to schedule workflows to run automatically at specified times or intervals, without any manual trigger or code push. It is defined using the schedule event in the workflow YAML file (example in practical)

```yaml
name: Scheduled Workflow

on:
  schedule:
    - cron: '0 0 * * *'   # Runs every day at midnight UTC

jobs:
  run-daily:
    runs-on: ubuntu-latest
    steps:
      - name: Say Hello
        run: echo "Hello, world!"
```

Uses **schedule** to setup cron expression

Workflow Dispatch

Workflow dispatch is an event trigger in GitHub Actions that allows you to manually trigger a workflow from the GitHub UI or via the API (example in practical)

```yaml
name: Manual trigger
on:
  workflow_dispatch:
    inputs:
      name:
        description: "Who to greet"
        default: "World"
```

- Uses **workflow_dispatch** for manual trigger. The code needs to merged to main before it can be manually triggered.
- This example uses input from user as well.

Best Practice – Part 1

1. **Maintain One Workflow Per Trigger Condition:** Each unique triggering event should have its own workflow. It should be noted that “push to dev” and “push to release” are different events, even though they both trigger on pushes

2. **Name (Entry) Workflows After The Trigger:** For not reusable workflows, name the file after the event which triggers it. For example:

   # name: "On Pull Request” – Good

   # name: "CI Pipeline” - Bad

3. **Use Dedicated Repositories for Actions:** Actions should be developed in a repository dedicated to the action. It is recommended by GitHub and provides versioning, tracking, and releasing the action like other software

Best Practices – Part 2

1. **Store Actions in .github/actions:** This only applies if you are ignoring Best Practice - 2. If an action is present in a repository with other content such as application code) then the action should live in the *.github/actions/<name_of_action>* directory

2. **Store Reusable Workflows Separate From Application Code:** Reusable workflows should be stored in a repository which does not include application code. Ideally reusable workflows should be stored in a centralized repository

3. **Version CI/CD Components:** All CI/CD components including workflows, reusable workflows, actions, and composite actions should be managed by version control.

Interested to know more?

Practical Example

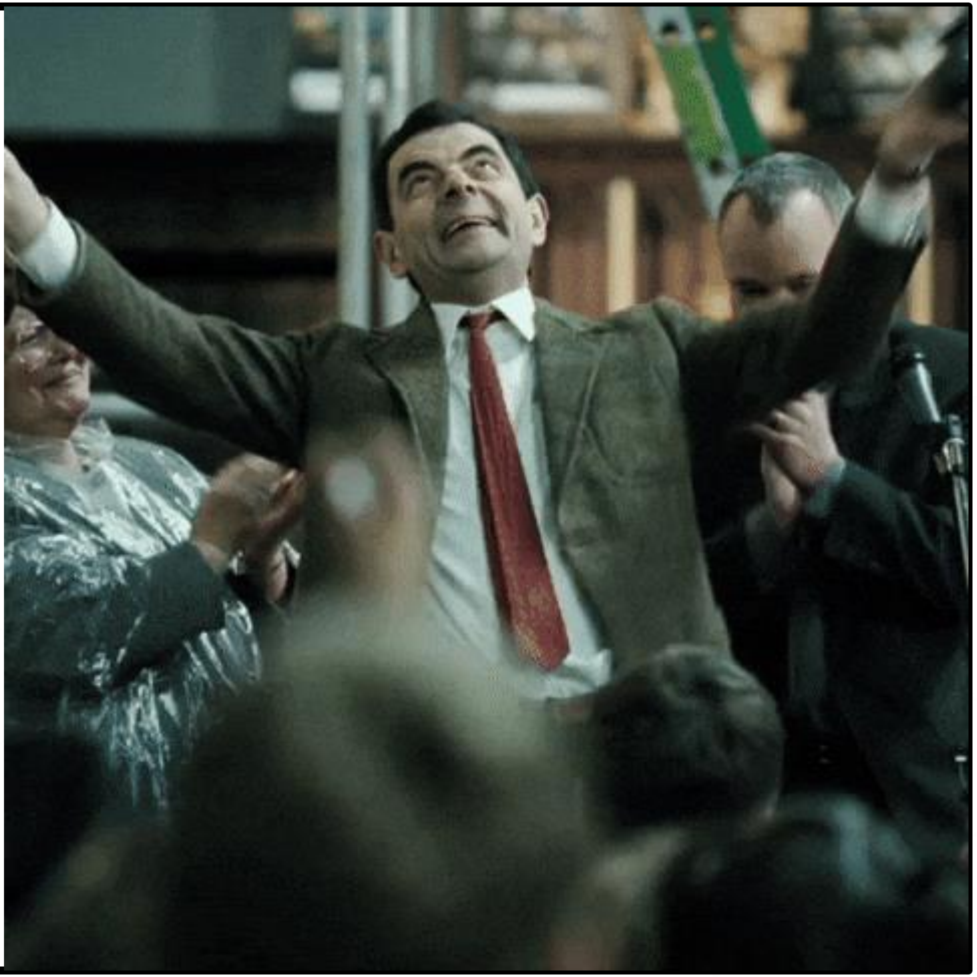
- A workflow with one job, multiple event triggers, and Cronjob – **Basic**
- A workflow with Workflow Dispatch– **Medium**
- A workflow with Docker Container, JavaScript, and Composite Actions – **Advance**

Access the link to the GitHub repository for practical examples.

Competition

- *Link to the **Kahoot** Competition*

- **Prizes** for 1st , 2nd and 3rd winners!



Bonus - Hidden

Register for GitHub Education program with your university email and get a free voucher for GitHub Foundation certification.

Preparation material for GitHub Action certification from Microsoft.

GitHub Actions - Advance

Osama Tahir

Software Engineer at ASML

Agenda

- ❖ **Jekins to GitHub Actions Migration**
  - o Introduction – Understanding Pipelines
  - o Examples
  - o Jenkins Pipeline
  - o GitHub Action Pipeline

- ❖ **Setting Multi-Project CI/CD using GitHub Actions for Enterprise**
  - o Simple Workflow
  - o Shared Versus Dedicated Repository
  - o CI/CD Repository Architecture

- Before Q/A

- ❖ **Best Practices for Reusable Workflows in GitHub Actions**
  - o Best Practices

- o **Usage and Performances Metrics in GitHub Actions**
  - o GitHub Actions Metrics
  - o Performance and Usage

- After Q/A

Jenkins to GitHub Actions Migration

Introduction - Understanding Pipelines

There is increasing trend of adopting cloud-native and integrated DevOps tools. There are key drivers for migration, such as scalability and reduced maintenance overhead. When building pipelines, understand the following:

- Stages and steps
- Environment variables
- Dependencies
- Artifacts
- External integrations

Jenkins Pipeline

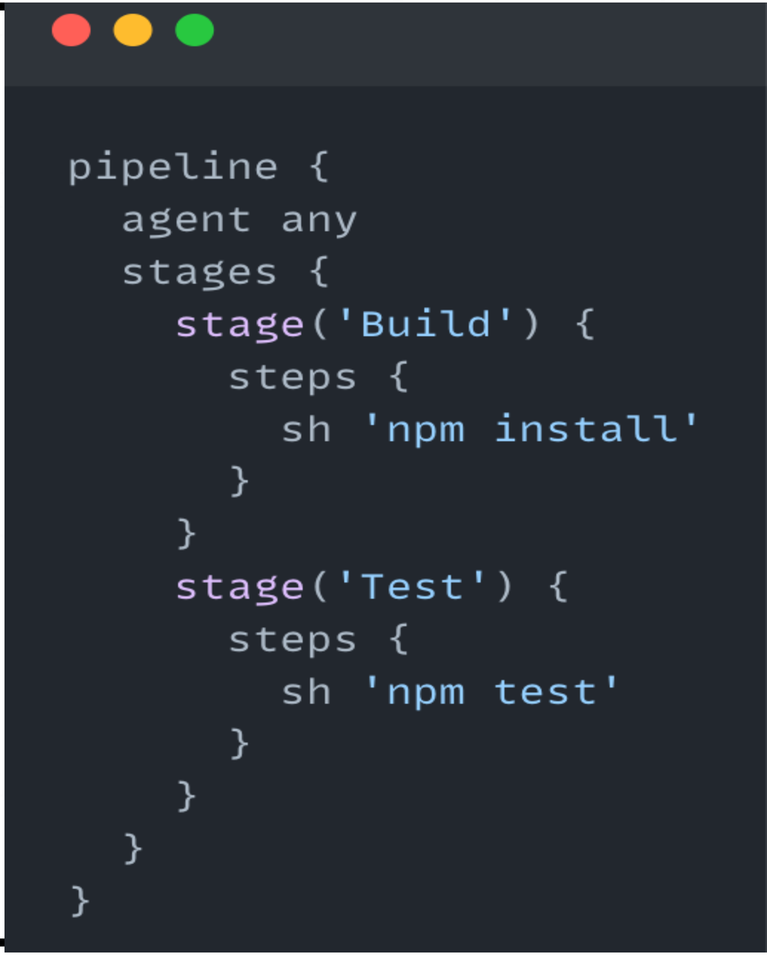
Use case: Self-hosted or enterprise-grade CI/CD workflows, offering more control and flexibility.

Syntax: Groovy-based DSL (Domain Specific Language), allows for complex pipelines.

Typical Features:

- Integration with hundreds of plugins
- Parallel execution and highly customizable.
- Supports distributed builds across multiple agents.

Example Best for: Complex pipelines that require conditional stages, sophisticated artifact management, integration with legacy systems, and scalable builds.



```
pipeline {
  agent any
  stages {
    stage('Build') {
      steps {
        sh 'npm install'
      }
    }
    stage('Test') {
      steps {
        sh 'npm test'
      }
    }
  }
}
```

GitHub Actions Pipeline

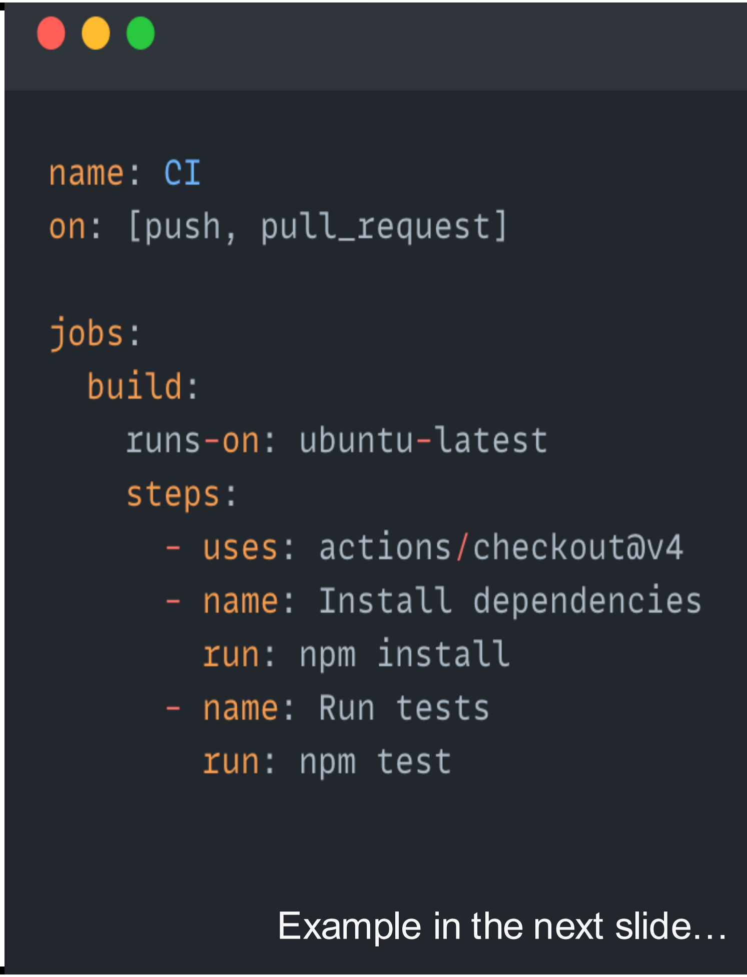
Use case: CI/CD workflows hosted directly on GitHub, better for projects tightly integrated with GitHub.

Syntax: YAML-based, simple, and easy to understand.

Typical Actions:

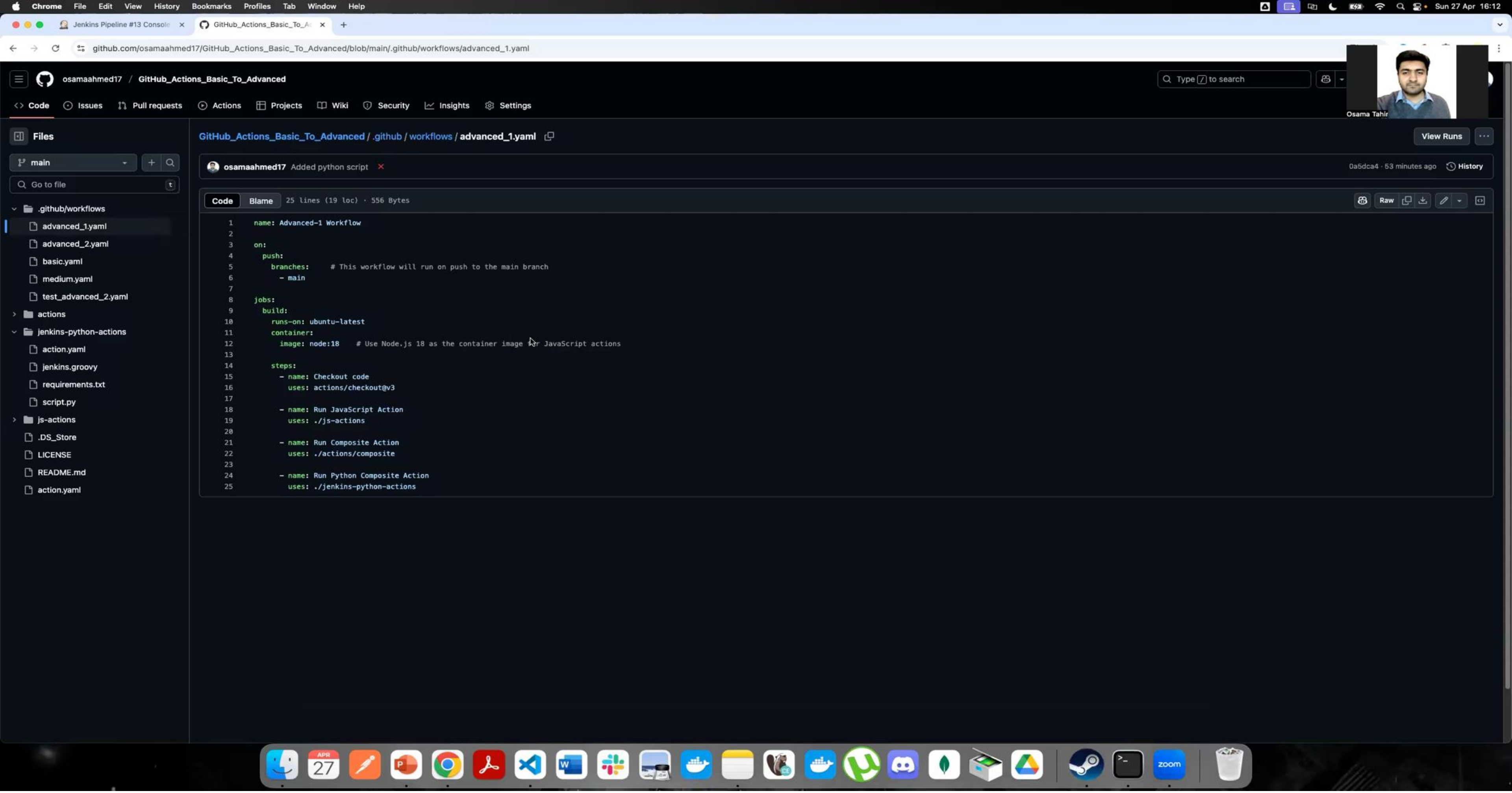
- **docker/build-push-action** for Docker
- **actions/checkout** to pull repo
- **actions/cache** to speed up workflows
- **actions/setup-node**, **actions/setup-python** for language-specific environments

Example Best for: Continuous integration of web apps, testing, Docker image building, deployment on cloud platforms.



```yaml
name: CI
on: [push, pull_request]

jobs:
  build:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - name: Install dependencies
        run: npm install
      - name: Run tests
        run: npm test
```

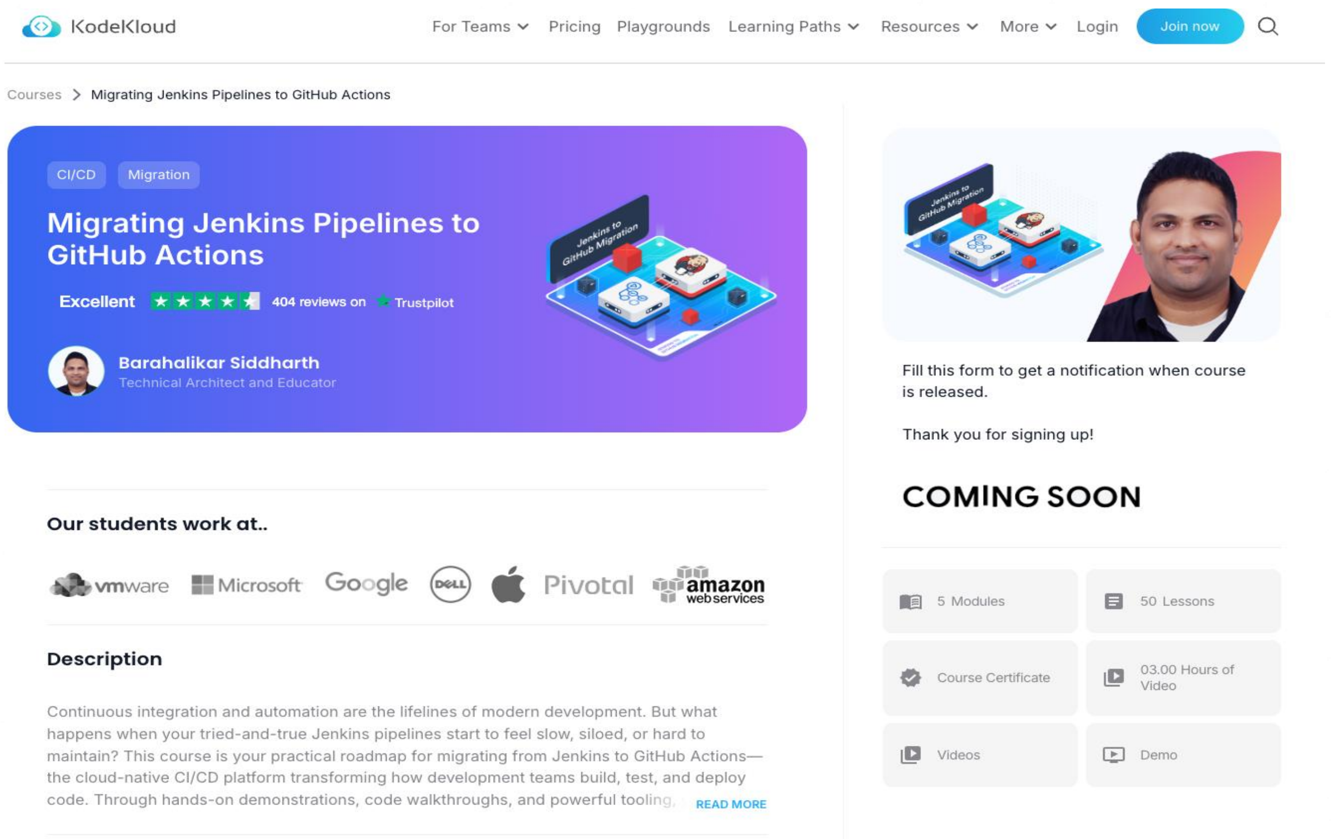
Example in the next slide…



```yaml
name: Advanced-1 Workflow

on:
  push:
    branches:     # This workflow will run on push to the main branch
      - main

jobs:
  build:
    runs-on: ubuntu-latest
    container:
      image: node:18    # Use Node.js 18 as the container image for JavaScript actions

    steps:
      - name: Checkout code
        uses: actions/checkout@v3

      - name: Run JavaScript Action
        uses: ./js-actions

      - name: Run Composite Action
        uses: ./actions/composite

      - name: Run Python Composite Action
        uses: ./jenkins-python-actions
```

The GitHub Action and Jenkin file shown in this video, can be found here.

Course: Migrating Jenkins Pipelines to GitHub Actions



Courses > Migrating Jenkins Pipelines to GitHub Actions

CI/CD Migration

Migrating Jenkins Pipelines to GitHub Actions

Excellent ★★★★★ 404 reviews on Trustpilot

Barahalikar Siddharth
Technical Architect and Educator

Fill this form to get a notification when course is released.

Thank you for signing up!

COMING SOON

Our students work at..

vmware Microsoft Google Dell Pivotal amazon web services

Description

Continuous integration and automation are the lifelines of modern development. But what happens when your tried-and-true Jenkins pipelines start to feel slow, siloed, or hard to maintain? This course is your practical roadmap for migrating from Jenkins to GitHub Actions—the cloud-native CI/CD platform transforming how development teams build, test, and deploy code. Through hands-on demonstrations, code walkthroughs, and powerful tooling, READ MORE

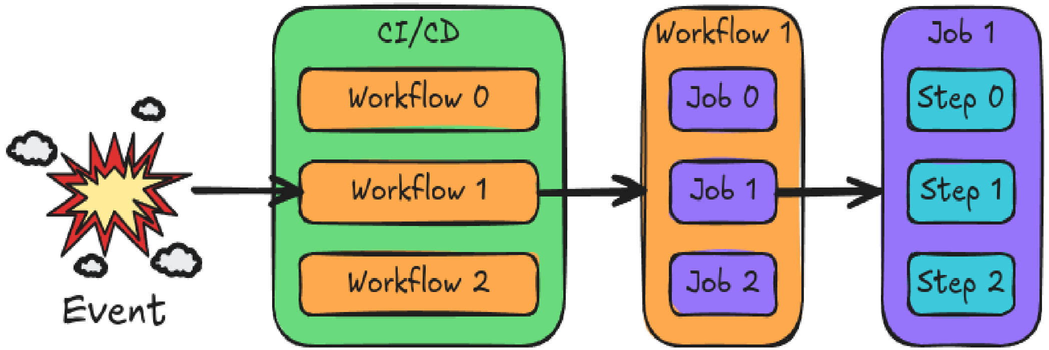
5 Modules
50 Lessons
Course Certificate
03.00 Hours of Video
Videos
Demo

Setting Multi-Project CI/CD using GitHub Actions for Enterprise

Simple Workflow

An entry workflow is triggered directly by a GitHub event. Must be stored inside the *.github/workflows/* directory of the repository. Main components includes:

- **Event**: What triggers the workflow
- **Workflow**: Set of instructions that define the automation
- **Job**: A collection of steps, run on a runner
- **Step**: An individual task inside a job
- **Runner**: GitHub-hosted or self-hosted machine executing the jobs



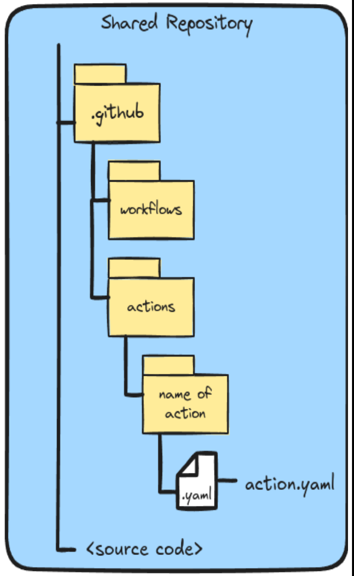
Shared Versus Dedicated Repository

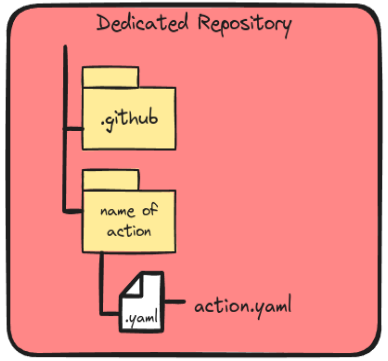
Shared Repository

- Actions are stored alongside application code and CI/CD workflows
- Actions live under *.github/actions/*, each inside its own subfolder
- Best when actions are closely tied to the repository's workflows or application

Dedicated Repository

- The entire repository is dedicated to hosting an action
- Action lives directly in a subfolder under the root of the repository
- Ideal for sharing actions across multiple repositories or publishing actions publicly.

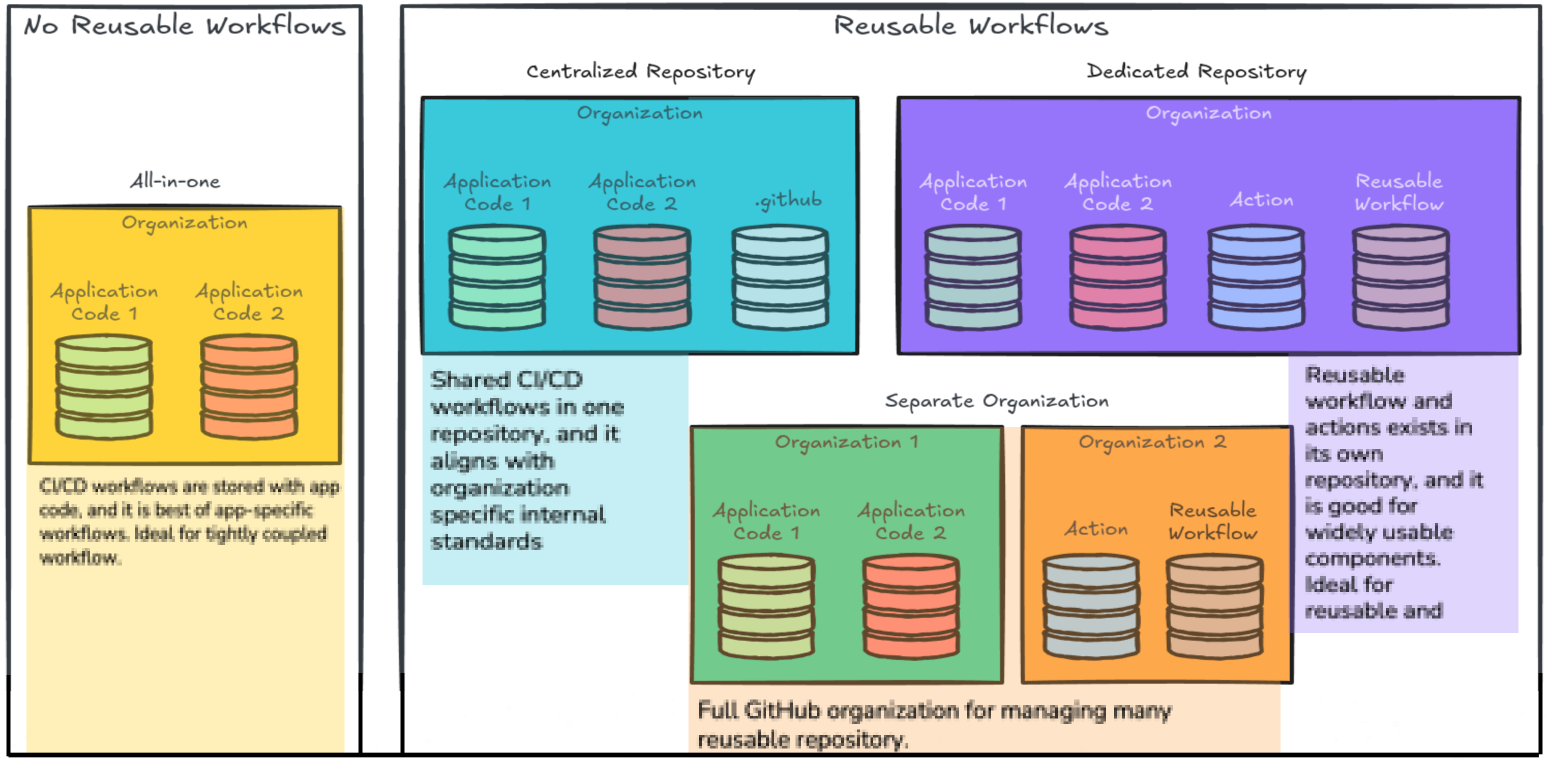




CI/CD Repository Architecture

How CI/CD components such as workflows and actions are distributed across repositories in a GitHub organization?

Choices impacts **reusability**, **maintainability**, and **scaling**. There are different architectures:



CI/CD workflows are stored with app code, and it is best of app-specific workflows. Ideal for tightly coupled workflow.

Shared CI/CD workflows in one repository, and it aligns with organization specific internal standards

Reusable workflow and actions exists in its own repository, and it is good for widely usable components. Ideal for reusable and

Full GitHub organization for managing many reusable repository.

Best Practices for Reusable Workflows in GitHub Actions

Composite Actions - Structure

Composite actions allow you to collect a series of workflow job steps into a single action which you can then run as a single job step in multiple workflows (practical example)

```yaml
# .github/actions/python-test-action/action.yml
name: Python Test Action
description: Runs Python unit tests
runs:
  using: "composite"
  steps:
    - name: Checkout code
      uses: actions/checkout@v2
    - name: Run Python tests
      run: python -m unittest discover
```

Composite Action

```yaml
# .github/workflows/call-composite-action.yml
name: Call Python Test Action

on: push

jobs:
  run-tests:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout code
        uses: actions/checkout@v2

      - name: Run Python Test Action
        uses: org-name/repo-name/.github/actions/python-test-action@main
```

Calling a Composite Action

Reusable Workflows - Structure

Reusable workflows can be centrally maintained, in one location, but used in many repositories across your organization

```yaml
# .github/workflows/reusable-workflow.yml
name: Reusable CI Workflow
on: workflow_call

jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout code
        uses: actions/checkout@v2
      - name: Run Python tests
        run: python -m unittest discover
```

Reusable Workflow

```yaml
# .github/workflows/call-reusable-workflow.yml
name: Call Reusable CI Workflow

on: push

jobs:
  call-workflow:
    uses: org-name/repo-
name/.github/workflows/reusable-workflow.yml@main
```

Calling Reusable Workflow

Best Practice – Part 1

Parameterize Your Workflow
Parameterize your workflows with default values and clear descriptions. It ensures smooth execution and improves team understanding.

Workflow Documentation
Document your workflows clearly with usage instructions, inputs, outputs, and comments. It make it easy for everyone to understand and use.

Use Composite Actions
Use composite actions to simplify workflows. It groups overlapping steps into a single, reusable action for better organization and maintainability.

Consider a Dedicated Workflows Repository
Set up a dedicated workflows repository to standardize, version, and streamline reusable workflows across your organization.

Best Practice – Part 2

Version Your Reusable Workflows

Version reusable workflows by creating releases. This allows teams to reference clean and readable tags.

Test Your Reusable Workflows

Test reusable workflows by passing input arguments and verifying outputs. Use matrix strategies for broader and efficient coverage. Check here

Follow Naming Conventions

Use clear, descriptive names for workflows, jobs, and steps to make their purpose easily understandable and improve usability.

My suggestion: Don’t let the fire burn, follow best practices!

Usage and Performances Metrics in GitHub Action

GitHub Actions Metrics

GitHub Actions metrics provide insights into how your workflows and jobs are performing at the organization and repository levels. There are two types of metrics to analyse workflows:

- **GitHub Actions usage metrics:** Track minutes of workflows and jobs consume. Useful for identifying high-usage workflows or repositories.

- **GitHub Actions performance metrics:** Focus on the efficiency and reliability of workflows and jobs. Monitor indicators like job run times, queue times, and failure rates

Performance and Usage

Gain insights into the efficiency, reliability, and resource consumption of your workflows across the organization.

- **Workflows**: Analyse average run time, job failures, and Actions minutes usage. Identify optimization opportunities like refactoring or using larger runners.

- **Jobs**: Monitor average run time, queue time, failures, and identify the most resource-intensive jobs and their runtime environments.

- **Repositories**: Get a high-level snapshot of each repository's performance and total Actions minutes usage.

- **Runtime OS**: Understand how runners perform across operating systems and which OS types are most used.

References, Practical, and Certification

- https://docs.github.com/en/actions/migrating-to-github-actions/manually-migrating-to-github-actions/migrating-from-jenkins-to-github-actions - key-differences
- https://multiprojectdevops.github.io/tutorials/1_github_actions/
- https://earthly.dev/blog/github-actions-reusable-workflows/
- https://docs.github.com/en/actions/administering-github-actions/viewing-github-actions-metrics

❖ **Practical Implementation:**
https://github.com/osamaahmed17/GitHub_Actions_Basic_To_Advanced

Preparation material for GitHub Advanced Security certification from Microsoft.:
https://learn.microsoft.com/en-us/training/paths/github-advanced-security/?wt.mc_id=github_inproduct_ghas_mslearn_ghcertregistration

Question and Feedback

Connect me on
LinkedIn

osamaahmed17

For
Feedback