GitHub Actions is a continuous integration and continuous delivery (CI/CD) platform that allows you to automate your build, test, and deployment pipeline

**Types of Runners**

GitHub-Hosted Runners:

Managed by GitHub: GitHub provides and maintains the environment.

Pre-installed Tools: Includes popular tools, languages, and frameworks (like Python, Node.js, Docker, etc.).

OS Options: Available for Ubuntu, macOS, and Windows.

Scalability: Automatically scales to handle workflows.

Use Case: Best for general workloads that don't require custom configurations.

Self-Hosted Runners:

User-Managed: Installed and maintained on your own hardware or infrastructure.

Customizable: You can configure them with proprietary tools, unique hardware, or specialized software.

Security: You control the security and access to the environment.

Cost: Requires user investment in infrastructure.

Use Case: Ideal for specific needs, such as accessing on-premises systems or using custom configurations.

===================================================================================================================================================

name: Build, Test, Analyze, and Publish Artifacts

on:

push:

branches:

- main # Trigger workflow on pushes to the 'main' branch

jobs:

build-test-analyze-publish:

runs-on: ubuntu-latest

steps:

# Step 1: Check out the code

- name: Checkout Code

uses: actions/checkout@v3

# Step 2: Set up JFrog CLI

- name: Set up JFrog CLI

uses: jfrog/setup-jfrog-cli@v3

with:

version: 2.41.0 # Specify the desired JFrog CLI version

# Step 3: Configure JFrog CLI

- name: Configure JFrog CLI

run: |

jfrog config add artifactory-server \

--url=https://<your-artifactory-url> \

--user=${{ secrets.ARTIFACTORY\_USER }} \

--password=${{ secrets.ARTIFACTORY\_PASSWORD }}

# Step 4: Set up Java

- name: Set up Java

uses: actions/setup-java@v3

with:

distribution: 'temurin'

java-version: '17'

# Step 5: Build and Test the Project

- name: Build and Test

run: mvn clean verify

# Step 6: SonarQube Analysis

- name: SonarQube Scan

env:

SONAR\_TOKEN: ${{ secrets.SONAR\_TOKEN }} # SonarQube token stored as a secret

run: |

mvn sonar:sonar \

-Dsonar.projectKey=<your-project-key> \

-Dsonar.host.url=https://<your-sonarqube-url> \

-Dsonar.login=${{ secrets.SONAR\_TOKEN }}

# Step 7: Upload Artifacts to Artifactory

- name: Upload Artifacts

if: success() # Ensures this step only runs if previous steps succeeded

run: |

jfrog rt u \

target/\*.jar \

generic-local/my-app/ # Replace with your Artifactory repository and path

Checkout Code: Pulls the repository code to the GitHub runner using actions/checkout.

Set Up JFrog CLI: Installs the JFrog CLI tool for interacting with Artifactory.

Configure JFrog CLI: Configures JFrog CLI with your Artifactory URL and credentials (stored as GitHub Secrets).

Set Up Java: Installs the required Java version (e.g., Java 17) for building and testing the project.

Build and Test: Runs mvn clean verify to compile the project and execute tests.

SonarQube Analysis: Performs static code analysis using SonarQube with the mvn sonar:sonar command, authenticated via a SonarQube token.

Upload Artifacts: Pushes the built .jar files to Artifactory using JFrog CLI, but only if all previous steps succeed (if: success()).

=================================================================================================================================================

name: Comprehensive Workflow Triggers

1. **Trigger on Push to Specific Branches or Paths**

This workflow triggers when changes are pushed to the main branch, any branch matching the release/\* pattern, or when files in the src/ directory are modified, excluding changes in the docs/ directory.

yaml

on:

push:

branches:

- main

- release/\*

paths:

- src/\*\*

- '!docs/\*\*'

Example Scenarios:

Pushing a commit to main that modifies src/index.js will trigger the workflow.

Pushing a commit to release/v1.0 that modifies src/app.js will trigger the workflow.

Pushing a commit to main that only modifies docs/readme.md will not trigger the workflow.

2. **Trigger on Pull Request Events**

This workflow triggers when a pull request targets the main branch and includes changes in the src/ directory.

yaml

on:

pull\_request:

branches:

- main

paths:

- src/\*\*

Example Scenarios:

A pull request to main that modifies src/app.js will trigger the workflow.

A pull request to main that only modifies docs/readme.md will not trigger the workflow.

3. **Trigger on a Schedule**

This workflow runs automatically every Monday at 00:00 UTC.

yaml

on:

schedule:

- cron: '0 0 \* \* 1'

Example Scenario:

The workflow executes at 00:00 UTC every Monday, which is 05:30 AM IST in Bengaluru, India.

4. **Trigger Manually via the GitHub UI (workflow\_dispatch)**

This workflow can be triggered manually from the GitHub Actions tab, allowing the user to select an environment.

yaml

on:

workflow\_dispatch:

inputs:

environment:

description: 'Select environment'

required: true

default: 'production'

type: choice

options:

- development

- staging

- production

Example Scenario:

A user manually triggers the workflow and selects the staging environment from the provided options.

5. **Trigger on Repository Dispatch**

This workflow triggers when a custom event named custom-event is dispatched to the repository.

yaml

on:

repository\_dispatch:

types:

- custom-event

Example Scenario:

An external service dispatches a custom-event to the repository via the GitHub API, triggering the workflow.

6. **Trigger on Release Events**

This workflow runs when a release is created or published.

yaml

on:

release:

types:

- created

- published

Example Scenarios:

Creating a new release in the repository triggers the workflow.

Publishing a draft release triggers the workflow.

7**. Trigger on Issue Events**

This workflow runs when an issue is opened or closed.

yaml

on:

issues:

types:

- opened

- closed

Example Scenarios:

Opening a new issue triggers the workflow.

Closing an existing issue triggers the workflow.

8. **Trigger on Comments (issue\_comment)**

This workflow runs when a new comment is added to an issue or pull request.

yaml

on:

issue\_comment:

types:

- created

Example Scenario:

Adding a comment to a pull request triggers the workflow.

9. **Trigger on a Specific Tag Push**

This workflow triggers when a tag matching the pattern v\*.\*.\* (e.g., v1.2.3) is pushed.

yaml

on:

push:

tags:

- 'v\*.\*.\*'

Example Scenario:

Pushing a tag v1.2.3 to the repository triggers the workflow.

10. **Trigger on Deletion Events**

This workflow runs when a branch or tag is deleted.

yaml

on:

delete:

branches: true

tags: true

Example Scenarios:

Deleting a branch triggers the workflow.

Deleting a tag triggers the workflow.

11. **Trigger on Forked Repository Events**

This workflow runs when the repository is forked.

yaml

on:

fork: {}

Example Scenario:

A user forks the repository, triggering the workflow.

12. **Trigger on Starred Repository Events**

This workflow runs when someone stars the repository.

yaml

on:

star: {}

Example Scenario:

A user stars the repository, triggering the workflow.

13. **Trigger on Watch Events**

This workflow runs when someone starts watching the repository.

yaml

on:

watch: {}

Example Scenario:

A user starts watching the repository, triggering the workflow.

jobs:

example\_job:

runs-on: ubuntu-latest

steps:

- name: Display Trigger Context

run: echo "Workflow triggered by ${{ github.event\_name }}"

**Push and Pull Request**: Specify branches and paths to limit when workflows run.

**Schedule**: Use CRON syntax for time-based triggers.

**Workflow Dispatch**: Enable manual execution with optional inputs.

**Repository Dispatch**: Trigger workflows programmatically via GitHub API.

**Release and Tags**: Respond to events like publishing a release or pushing a tag.

**Issues and Comments**: Automate workflows for issue and PR activity.

**Deletion, Fork, Star, and Watch**: Handle repository-level events like deletions or interactions.

=====================================================================================================================================================

Conditional Job Execution

**Execute Based on Branch**

**Execute Based on Event**

**Execute Based on Matrix Values**

**Execute Based on the Outcome of Another Job**

**Execute Based on Workflow Input**

**Execute Based on Environment Variables**

name: Conditional Execution Pipeline

# Trigger events

on:

push:

branches:

- main # Run pipeline when changes are pushed to the main branch

- develop # Run pipeline for the develop branch as well

pull\_request:

branches:

- main # Run pipeline when PR is created targeting 'main'

workflow\_dispatch: # Allows manual trigger with input variables

inputs:

deploy\_env:

description: 'Environment to deploy to'

required: true

default: 'staging'

jobs:

# Step 1: Checkout Code (This job is needed for every job)

checkout\_code:

runs-on: ubuntu-latest

steps:

- name: Checkout code from repository

uses: actions/checkout@v2

# Step 2: Build Job - Always Runs (Strategy 1: Always for Any Event)

build:

runs-on: ubuntu-latest

needs: checkout\_code

steps:

- name: Set up Java 17

uses: actions/setup-java@v2

with:

java-version: '17'

distribution: 'temurin'

- name: Build with Maven

run: mvn clean install

# Step 3: Test Job - Run if Build is Successful (Strategy 2: Outcome-based Execution)

test:

runs-on: ubuntu-latest

needs: build

if: success() # Runs only if the build job succeeds

steps:

- name: Run tests

run: mvn test

# Step 4: Deploy Job Based on Branch (Strategy 3: Branch-based Execution)

deploy\_branch:

runs-on: ubuntu-latest

needs: test

if: github.ref == 'refs/heads/main' # Only runs on 'main' branch

steps:

- name: Deploy to Production (Main branch only)

run: echo "Deploying to production because we're on the main branch."

# Step 5: Deploy Job Based on Event Type (Strategy 4: Event-based Execution)

deploy\_event:

runs-on: ubuntu-latest

needs: test

if: github.event\_name == 'push' # Only runs for push events, not PR

steps:

- name: Deploy on Push Event

run: echo "Deploying because it's a push event, not a pull request."

# Step 6: Deploy Job Using Matrix (Strategy 5: Matrix Execution for Multiple Environments)

deploy\_matrix:

runs-on: ubuntu-latest

needs: test

strategy:

matrix:

environment: ['staging', 'production'] # Matrix to deploy to both environments

steps:

- name: Deploy to ${{ matrix.environment }}

run: echo "Deploying to ${{ matrix.environment }}"

# Step 7: Deploy Based on Workflow Input (Strategy 6: Input-based Execution)

deploy\_input:

runs-on: ubuntu-latest

needs: test

if: ${{ github.event.inputs.deploy\_env == 'production' }} # Only if user input is 'production'

steps:

- name: Deploy based on Input

run: echo "Deploying to ${GITHUB\_ENV} environment based on input."

# Step 8: Deploy Based on Environment Variable (Strategy 7: Env Variable-based Execution)

deploy\_env\_var:

runs-on: ubuntu-latest

needs: test

if: ${{ env.DEPLOY\_ENV == 'production' }} # Only run if DEPLOY\_ENV is 'production'

steps:

- name: Deploy based on ENV variable

run: echo "Deploying to production because DEPLOY\_ENV is set to production."

# Step 9: Always Run Cleanup (Strategy 8: Always Executed)

cleanup:

runs-on: ubuntu-latest

if: always() # This job runs regardless of the previous jobs' results (success/failure)

steps:

- name: Clean up

run: echo "Cleaning up after all jobs are completed."

=====================================================================================================================================================

name: Complete Execution Strategy Example

on:

push:

branches: [main]

pull\_request:

branches: [main]

workflow\_dispatch:

inputs:

deploy:

description: 'Deploy the app'

required: true

default: 'false'

jobs:

# Parallel Execution (Independent Jobs)

build:

runs-on: ubuntu-latest

steps:

- run: echo "Building..."

test:

runs-on: ubuntu-latest

steps:

- run: echo "Testing..."

# Sequential Execution (Dependent Jobs)

deploy:

runs-on: ubuntu-latest

needs: test # Runs after 'test' job

steps:

- run: echo "Deploying..."

# Conditional Execution Based on Outcome

conditional\_test:

runs-on: ubuntu-latest

needs: build # Dependent on 'build' job

if: success() # Runs only if 'build' is successful

steps:

- run: echo "Conditional Test: Build successful, running tests..."

# Matrix Execution (Parallel with Different Configurations)

matrix\_test:

runs-on: ubuntu-latest

strategy:

matrix:

node-version: [12, 14]

steps:

- run: echo "Testing Node.js version ${{ matrix.node-version }}"

# Event-based Execution (Runs only on specific event types)

push\_event\_job:

runs-on: ubuntu-latest

if: github.event\_name == 'push'

steps:

- run: echo "Push Event: This runs on push events."

pr\_event\_job:

runs-on: ubuntu-latest

if: github.event\_name == 'pull\_request'

steps:

- run: echo "PR Event: This runs on pull request events."

# Execution Based on Workflow Input

deploy\_input\_job:

runs-on: ubuntu-latest

if: ${{ github.event.inputs.deploy == 'true' }}

steps:

- run: echo "Deploying based on input trigger."

--------------------------------------------------------------------------------------------------------------------------------------------------

Parallel Execution:

build and test jobs run independently in parallel.

Sequential Execution:

deploy job runs after the test job completes (using needs: test).

Conditional Execution Based on Outcome:

conditional\_test job runs only if the build job succeeds (using if: success()).

Matrix Execution:

matrix\_test job runs in parallel for different Node.js versions (12 and 14).

Event-based Execution:

push\_event\_job runs only on push events.

pr\_event\_job runs only on pull request events.

Execution Based on Workflow Input:

deploy\_input\_job runs only if the deploy input is set to true from the workflow\_dispatch event.

====================================================================================================================================================

Passing Outputs Between Jobs :

name: Passing Outputs Between Jobs Example

on:

push:

branches: [main]

jobs:

build:

runs-on: ubuntu-latest

outputs:

version: ${{ steps.set\_version.outputs.version }}

steps:

- name: Checkout code

uses: actions/checkout@v2

- name: Set version output

id: set\_version

run: |

echo "version=1.0.0" >> $GITHUB\_ENV

echo "::set-output name=version::1.0.0"

deploy:

runs-on: ubuntu-latest

needs: build # Wait for the 'build' job to finish

steps:

- name: Checkout code

uses: actions/checkout@v2

- name: Use version from build job

run: |

echo "Deploying version ${{ needs.build.outputs.version }}"

===================================================================================================================================================

name: Simple Script Execution

on:

push:

branches: [main]

jobs:

# First job that runs a simple script

run\_first\_script:

runs-on: ubuntu-latest

steps:

- name: Checkout Code

uses: actions/checkout@v2

- name: Run first script

run: |

echo "This is the first script"

echo "Hello from first script!" > first\_output.txt

# Second job that depends on the first job and runs another script

run\_second\_script:

runs-on: ubuntu-latest

needs: run\_first\_script # This job runs only after the first job finishes

steps:

- name: Checkout Code

uses: actions/checkout@v2

- name: Run second script

run: |

echo "This is the second script"

cat first\_output.txt

====================================================================================================================================================