**What is GitHub Actions?**

GitHub Actions is a CI/CD (Continuous Integration/Continuous Deployment) tool integrated directly into GitHub. It allows developers to automate their software development workflows, such as building, testing, and deploying code. GitHub Actions helps in automating tasks across the development lifecycle through custom workflows triggered by events like code pushes, pull requests, or scheduled intervals.

A diagram of a diagram

Description automatically generated

**Key Concepts**

**A diagram of a flowchart

AI-generated content may be incorrect.**

* **Workflow**: A YAML-defined automation process that is triggered by events. Each workflow is made up of one or more jobs, and each job contains steps.
* **Job**: A unit of work that contains one or more steps. Jobs run in a virtual environment (runner) and can run sequentially or in parallel.
* **Step**: Each individual task that makes up a job. Steps can include shell commands or action calls.
* **Event**: A trigger that starts a workflow. Events can include things like pushing code, creating a pull request, or scheduling a task.
* **Runner**: A runner in GitHub Actions is a server that executes your workflow's jobs. It's the environment where your workflow code runs, and it processes the individual steps defined in a job (e.g., running scripts, installing dependencies, and compiling code). Each job in a GitHub Actions workflow runs on a specific runner, and the runner provides the operating system, software, and hardware resources required to execute your jobs.

**Types of Runners in GitHub Actions**

There are two primary types of runners in GitHub Actions:

1. **GitHub-Hosted Runners**
2. **Self-Hosted Runners**

**1. GitHub-Hosted Runners**

GitHub provides pre-configured virtual environments for your workflows. These runners are managed entirely by GitHub, meaning GitHub takes care of updating, scaling, and maintaining the runners for you.

**Key Features of GitHub-Hosted Runners:**

* **Automatic Provisioning**: GitHub automatically provisions, configures, and tears down these runners as part of the workflow.
* **Pre-installed Software**: These runners come with a variety of tools, libraries, and programming environments pre-installed, such as Node.js, Python, Java, Docker, and more.
* **Multiple Operating Systems**: GitHub-hosted runners are available for Linux, Windows, and macOS operating systems.

**2. Self-Hosted Runners**

A **self-hosted runner** is a machine that you manage yourself. This machine can be on your local network, in the cloud, or in a data center, and it allows you to run jobs from your GitHub workflows.

**Key Features of Self-Hosted Runners:**

* **Full Control**: You have complete control over the environment and its configuration. You can install custom software, configure network settings, and adjust hardware as needed.
* **Persistence**: Unlike GitHub-hosted runners, self-hosted runners can maintain data between workflow runs since they aren't ephemeral by default.
* **Cost Efficiency**: For larger organizations with many workflows or for very custom environments, self-hosted runners can be more cost-effective, especially since you don’t rely on GitHub's hosted resources.

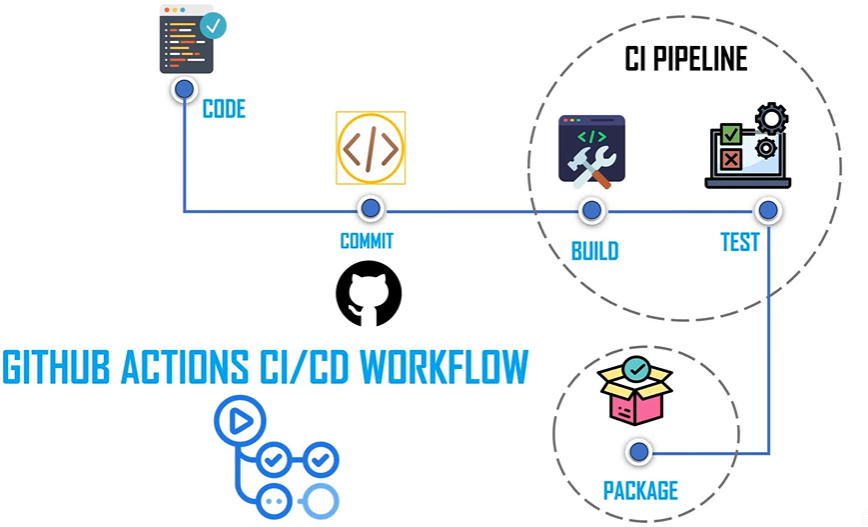
**Basic Workflow Structure**

A simple GitHub Actions workflow is written in a YAML file (.github/workflows/ directory). Below is an example of a basic structure:

A screen shot of a computer

Description automatically generated

**DEMO:**

****

Secret Management in GitHub Actions and Securing GitHub Actions

**Why Security is Important in CI/CD**

CI/CD systems are critical to the automation and efficiency of the software development lifecycle. However, they often handle sensitive information, such as API keys, credentials, and tokens, which makes them potential targets for security breaches. Proper security ensures that malicious actors cannot exploit workflows to gain access to your infrastructure, codebase, or sensitive data.

**Best Practices for Securing GitHub Actions**

* **Use Minimal Permissions**: Only give your workflows the permissions they need using the permissions key in your workflows.
* **Use Secrets**: Store sensitive data in GitHub Secrets and avoid hardcoding secrets in your workflows.
* **Use Workflow Triggers Carefully**: Limit which branches or repositories can trigger workflows to avoid untrusted code execution.
* **Pin Actions to Specific Versions**: Use specific versions of actions (actions/checkout@v2) to prevent the risk of malicious code in the latest versions.
* **Review Third-Party Actions**: Before using third-party actions, ensure they are trusted and have been reviewed for security.

**What are Secrets in GitHub?**

Secrets in GitHub are environment variables stored securely and encrypted. These secrets can contain sensitive data like API keys, passwords, or tokens. They are not visible in plain text and can be accessed only by workflows.

**Storing Secrets**

GitHub provides a secure way to store secrets in repositories or at the organization level.

* **Repository-Level Secrets**: You can add secrets for a single repository by navigating to the repository settings > Secrets.
* **Organization-Level Secrets**: These secrets are available across multiple repositories within an organization and are managed from the organization's settings.

**Accessing Secrets in Workflows**

Secrets are accessed in workflows via environment variables. Here's an example of how to use a secret in a workflow:

A screenshot of a computer program

Description automatically generated

In this example:

* **${{ secrets.DEPLOY\_API\_KEY }}**: This accesses the secret stored in the repository named DEPLOY\_API\_KEY.

**Viewing Workflow History**

GitHub Actions keeps a detailed history of all workflow runs, making it easy to monitor previous workflows, inspect logs, and identify issues.

To view workflow history:

1. Navigate to the **Actions** tab in your repository.
2. You'll see a list of workflows that have run, including their status (success, failure).
3. You can click on any workflow to see more details, including logs, timing, and outputs for each step of the workflow.

**Running Jobs in Parallel**

GitHub Actions supports running multiple jobs simultaneously (in parallel), which can speed up workflows by allowing independent tasks to execute concurrently.

A screenshot of a computer program

Description automatically generated

In this example, both the build and test jobs will run in parallel because there is no dependency between them. Each job runs in its own runner.

**Running Jobs Sequentially**

To run jobs sequentially, you can use the needs keyword in GitHub Actions to define dependencies between jobs. A job that "needs" another job will wait for the preceding job to finish before starting

A screen shot of a computer program

Description automatically generated

**Defining Reusable Workflows**

Reusable workflows are workflows that can be called from multiple repositories or workflows, improving consistency and reusability in your CI/CD pipelines. GitHub Actions allows you to define these reusable workflows in separate files, which can then be referenced in other workflows.

Reusable workflows are particularly useful when you have multiple repositories that require similar steps, such as building, testing, or deploying code.

A screenshot of a computer program

Description automatically generated

Now, you can call this reusable workflow from another workflow

jobs:

call-reusable-workflow:

**uses:** [**user/repo/.github/workflows/reusable-workflow.yml@main**](mailto:user/repo/.github/workflows/reusable-workflow.yml@main)

**What is Blue-Green Deployment?**

**Blue-Green Deployment** is a release management strategy that reduces downtime and risk during deployments. It involves running two identical production environments:

* **Blue**: The live, production environment currently serving users.
* **Green**: The environment where the new version of the application is deployed and tested.
* Once the new version (on the Green environment) is tested and verified, traffic is switched from the Blue environment to the Green environment. If any issues are encountered, the traffic can be switched back to Blue, making it easy to roll back.
* This method ensures zero downtime during deployment and allows for easy rollback in case of errors.

A screen shot of a computer

Description automatically generated

In this example, the deploy-green job deploys the new version to the green environment. Once that deployment is successful, the switch-traffic job switches the production traffic to the Green environment.

**Lab: Implement GitHub Actions**