What Is GitHub Actions?

GitHub Actions is a platform for continuous integration / continuous delivery (CI/CD). It enables you to automate build, testing, and deployment pipelines. It also lets you run arbitrary code on a specified repository when an event occurs. Actions uses code packages in Docker containers that run on GitHub servers. They are compatible with all programming languages to ensure you can run them on public clouds as well as local servers. This is part of an extensive series of guides about CI/CD.

GitHub Actions Architecture and Concepts

GitHub Actions allows you to configure your workflow to be triggered once a specific event occurs in the repository. For example, when an issue is created or when a pull request is opened.

A workflow contains one or several jobs running in parallel or sequential order. Each job runs inside a container or in a separate virtual machine (VM) runner. Additionally, each job includes one or several steps that run a predefined script or an action, a reusable extension that simplifies your workflow.

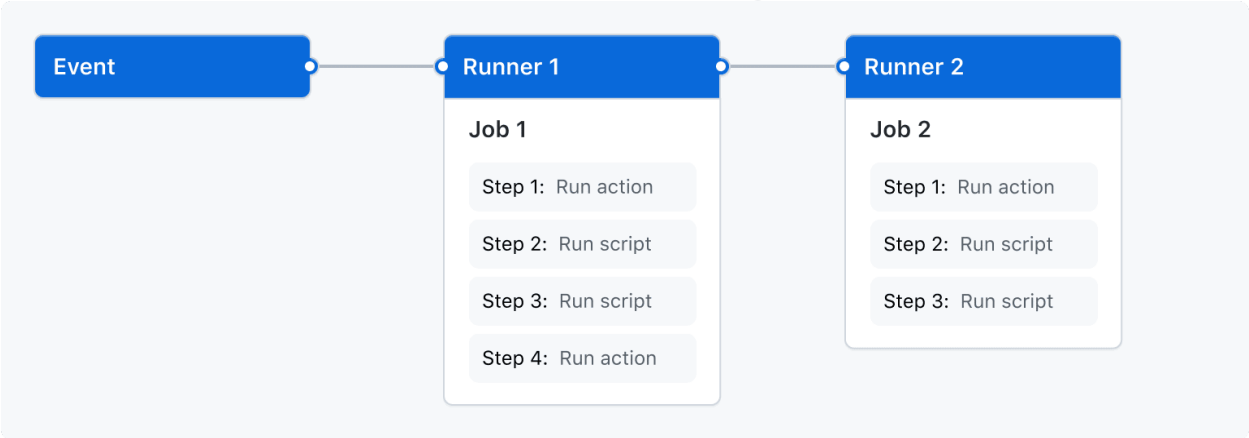

Image Source: Github Actions

GitHub Actions Workflows

A GitHub Actions workflow is an automated process that runs one or several jobs. You can configure workflows by defining a YAML file, which is checked into the repository. This file runs when:

- Triggered by a specific event in the repository
- Triggered manually
- At a predefined schedule

You can find workflow definitions in the .github/workflows directory of each repository. A repository can include several workflows, each performing different tasks. You can use, for example, one workflow for building and testing pull requests, another workflow for deploying the

application whenever a release is created, and an additional workflow for adding a label whenever a new issue is opened.

GitHub Actions Events

An event is a specific activity in a repository that triggers a workflow run. For example, activity can originate from GitHub when someone creates a pull request, opens an issue, or pushes a commit to a repository. You can also trigger a workflow run on a schedule, by posting to a REST API, or manually.

Workflow Triggers

A workflow trigger is an event that causes a workflow to run. Here are workflow trigger examples:

- Events occurring in the workflow's repository
- Events occurring outside of GitHub and trigger a repository_dispatch event on GitHub
- Manual
- Scheduled times

For instance, you can configure a workflow to run once a push is made to a default repository branch, once an issue is opened, and once a release is created.

GitHub Actions Jobs

A job in GitHub Actions is a series of workflow steps that run on a single runner. The steps can include actions to run or shell scripts for execution. Actions execute each step in order, as all steps in a job are interdependent. It executes every step on the same runner, enabling data sharing between steps. For instance, if you have a step to build an application, there might be a subsequent step to test the newly built application.

Jobs don't have dependencies by default, but you can configure jobs to be dependent on other jobs and run in sequence. Otherwise, the jobs will execute in parallel. If a job has a dependency on another job, it must wait for that job to complete before running. There could be several build jobs, each for a different architecture, without any dependencies—however, a packaging job would be dependent on these jobs. The non-dependent build jobs run simultaneously, but the packaging job can only run when all the build jobs are complete.

GitHub Actions Matrix

You can use a matrix strategy to run multiple related jobs based on the same job definition. Different combinations of variables in the job definition automatically create different job runs. For instance, a matrix might be useful for testing code on different operating systems or in different versions of a programming language.

Actions

Actions are custom applications for GitHub Actions that perform complex but repetitive tasks. You might use actions to avoid writing too much repetitive code in a workflow. An action could pull a repository from GitHub, set up authentication measures to the cloud provider, or configure the right toolchain for the development environment.

You can write an action from scratch or use an existing action available from the GitHub Marketplace in your workflow.

Runners

Runners are machines that have the GitHub Actions runner app installed. They wait for available jobs to execute. Once a runner picks up a job, it runs the actions specified by the job and reports the results to Github. You can host runners on your own server or machine or use GitHub-hosted runners.

Workflow Dependency Caching

A workflow run can reuse the outputs and dependencies from other runs. Dependency and package management tools like Maven, npm, Yarn, and Gradle keep local caches of all downloaded dependencies.

Each job on a GitHub-hosted runner starts in a fresh environment, so it is necessary to download the dependencies for every new job. These downloads can lengthen the runtime, increase network utilization, and increase costs. GitHub allows you to accelerate the process of recreating dependency files by caching frequently used files in a workflow.

You can cache job dependencies by using the cache action in GitHub. This action builds and restores caches identified using unique keys.

CI/CD with GitHub Actions

GitHub Action supports continuous integration and deployment.

Continuous Integration with GitHub Actions

Continuous integration is a software development practice that involves frequent commits to a shared code repository. GitHub Actions provides continuous integration (CI) workflows that can build code in the repository and execute tests. A workflow can run on a GitHub-hosted VM or a self-hosted machine.

You configure CI workflows to run in response to specific GitHub events—for instance, when a developer pushes new code to the repository. You can also use the repository dispatch webhook to schedule workflows or set triggers based on external events.

GitHub runs the CI tests, providing the results for all tests in a pull request. It lets you view any errors introduced to the branch by each change. Once a change has passed all the CI tests in the workflow, it is ready for review by team members, and you can merge it into the repository. If the commit fails a test, it is an indication of a breaking change.

When setting up continuous integration in the Git repository, GitHub will analyze your code and recommend specific CI workflows based on your repository's framework and language. For instance, GitHub suggests starter workflows that install the appropriate packages and run the tests (i.e., a Node.js package if you are using Node.js). You can use the GitHub-recommended

starter workflow as is, customize it to your needs, or create a new custom workflow to run the continuous integration tests.

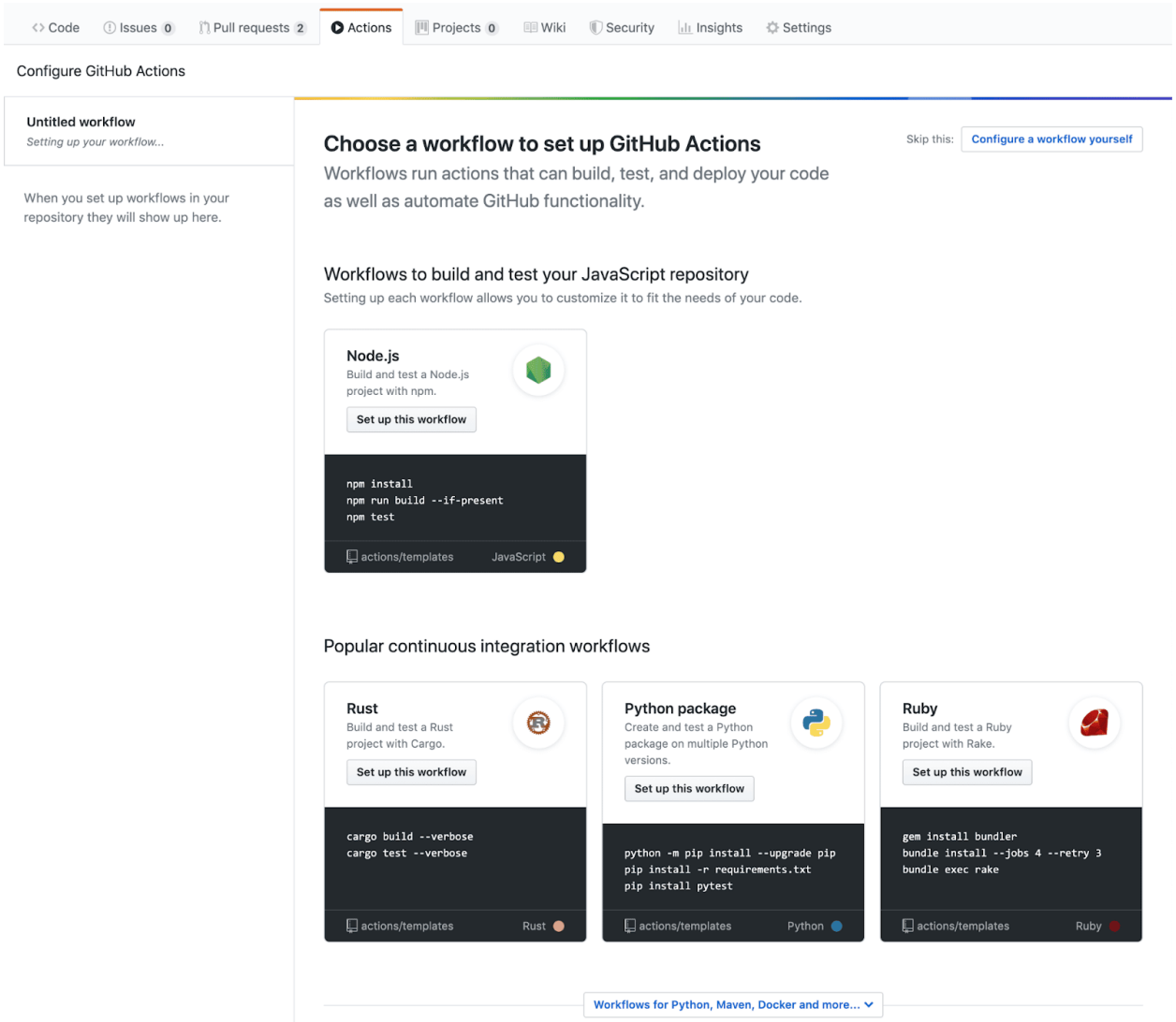

Image Source: GitHub Actions

GitHub Actions helps you set up a CI workflow for your development project and can create workflows for the entire SDLC. You could set up actions to package, deploy, and release a project.

Continuous Deployment with GitHub Actions

Continuous deployment is an extension of continuous integration and delivery that automates the process of publishing and deploying software releases. A CI/CD tool typically builds and tests the code automatically before deploying it as part of the continuous deployment process.

You can configure GitHub Actions workflows to deploy software products. A workflow can verify that the code functions as expected by building the code in the repository and running tests before deployment.

You can schedule CD workflows to run in response to specific GitHub events, such as a developer pushing code to the repository's default branch of your repository. Alternatively, workloads can run on set schedules, when triggered manually, or in response to external events.

GitHub Actions offers several features to help you control your deployments. Environments allow you to block jobs until they receive approval, restrict the branches that can trigger workflows, or restrict access to secrets. Concurrency lets you limit the CD pipeline to one in-progress and one pending deployment at a time.