

GitHub Actions Fundamentals



Presented by GitHub Professional Services



Our Agenda

Day 1

01

GitHub
Actions

Introduction

02

Actions
Workflow

Syntax

03

Actions
Secrets

Environments
and secrets

04

Manage
Actions

Managing
workflows &
Actions

05

Building
Actions

Build Actions &
Workflows



Our Agenda

Day 2

01

Migration

Migrate to
Actions

02

Runners

GitHub Actions
Runners

03

CI/CD

Action as CI/CD
workflows

Q/A



GitHub Actions

Introduction



What's GitHub Actions

GitHub Actions is a core CI/CD & automation feature that operates natively within an integrated DevOps platform

- ✓ Workflows stored as yml files
- ✓ Fully integrated with GitHub
- ✓ Respond to GitHub events
- ✓ Live logs and visualized workflow execution
- ✓ Community-powered and custom workflows
- ✓ GitHub-hosted, self-hosted, or Kubernetes hosted runners
- ✓ Built-in secrets and environment variables store



Actions & SDLC

Use cases across SDLC using GitHub Actions



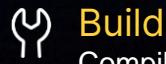
Plan

Tirage Issues and Project Boards using Actions



Code

Automates code builds, linting upon pull request creation



Build

Compile and build, Store artifacts



Test

Automation of the unit tests, integration tests



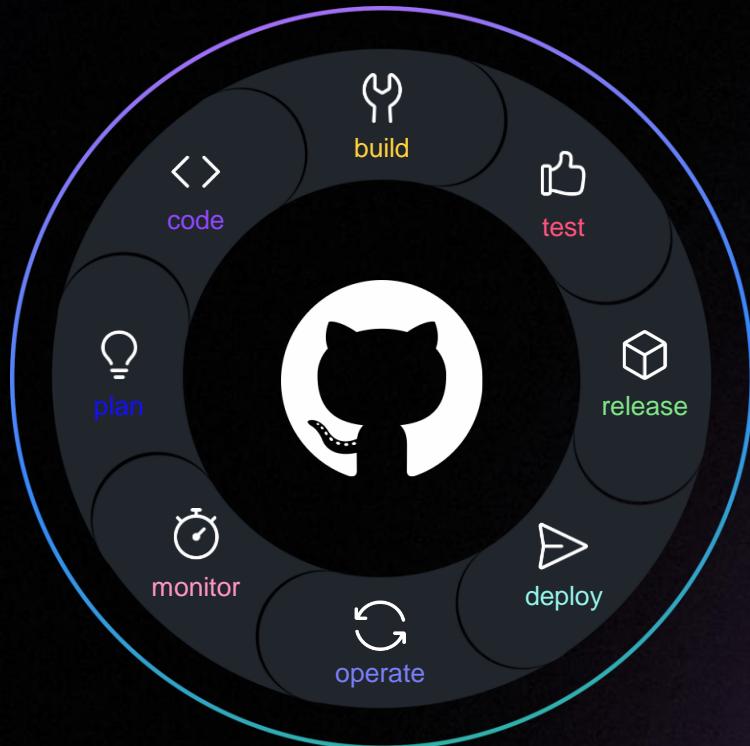
Release

Publish release, Release notes, Versioning, and Upload artifacts



Deploy

Deploy applications to your chosen environment





Automation accessible to every developer



Actions

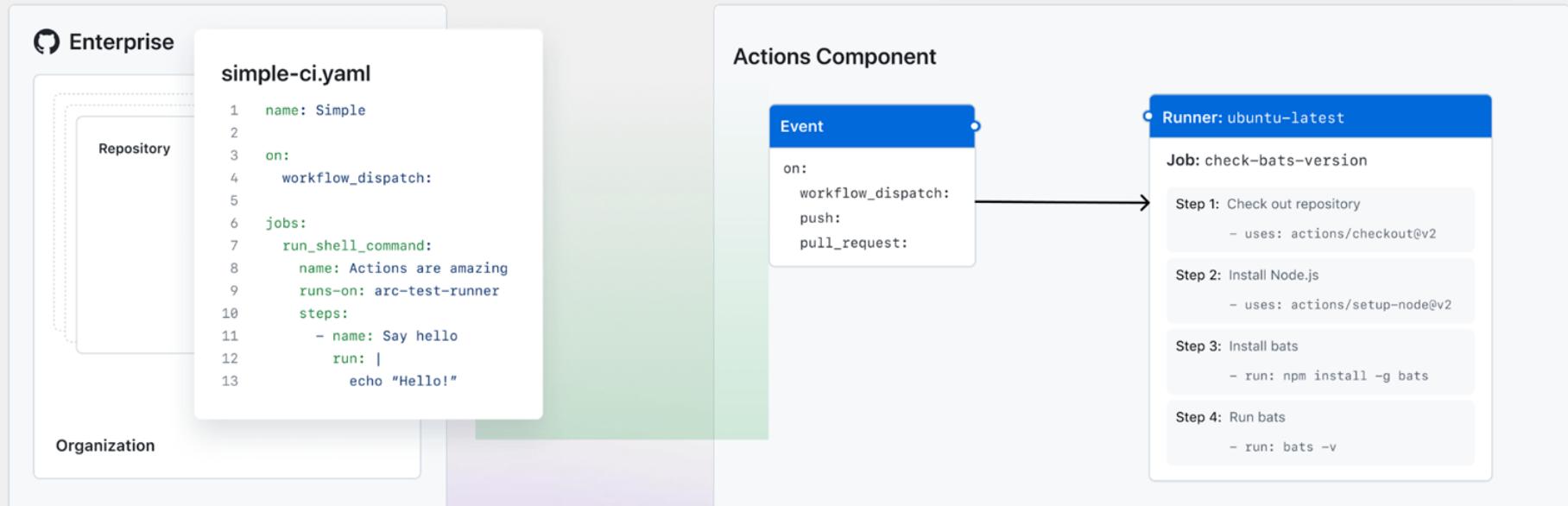
Enterprise-grade CI/CD that supports Windows, Linux, Mac

The screenshot shows the GitHub Actions interface. At the top, there are navigation links: Discussions, Actions (which is highlighted), Projects, Wiki, Security, Insights, and Settings. Below this, a table provides details about a workflow run: Manually triggered 2 minutes ago by octodemobot, Status In progress, Total duration -, and Artifacts -. The workflow file 'create_demo.yml' is shown, starting with 'on: workflow_dispatch'. The workflow consists of several steps: 'Initialize' (green circle), 'Configure Creation Steps' (green circle), 'Create Demo Repository...' (yellow circle, currently executing, with a tooltip 'Deploying to octodemo/testing-provisioning'), 'Register Demo Deployment S...' (grey circle), 'Update tracking issue' (grey circle), 'Matrix: Configure secrets' (grey circle), and 'Waiting for pending jobs' (grey circle). The status bar at the bottom indicates the workflow is in progress.

- ✓ Marketplace of 20,000+ Actions by our community
- ✓ Any operating system, cloud and on-prem
- ✓ Natively integrated into GitHub workflows
- ✓ Friction-free service or self-hosted runners
- ✓ Integrates Azure Pipelines & any other CI/CD
- ✓ DRY with Reusable Workflows
- ✓ API for viewing cache usage



GitHub Actions Key Components





Actions Workflow Syntax



Understanding the workflow file

The **name of the workflow** as it will appear in the "Actions" tab of the GitHub repository

The **name for workflow runs** generated from the workflow, which will appear in the list of workflow runs on your repository's "Actions" tab

Specifies **the trigger for this workflow**.

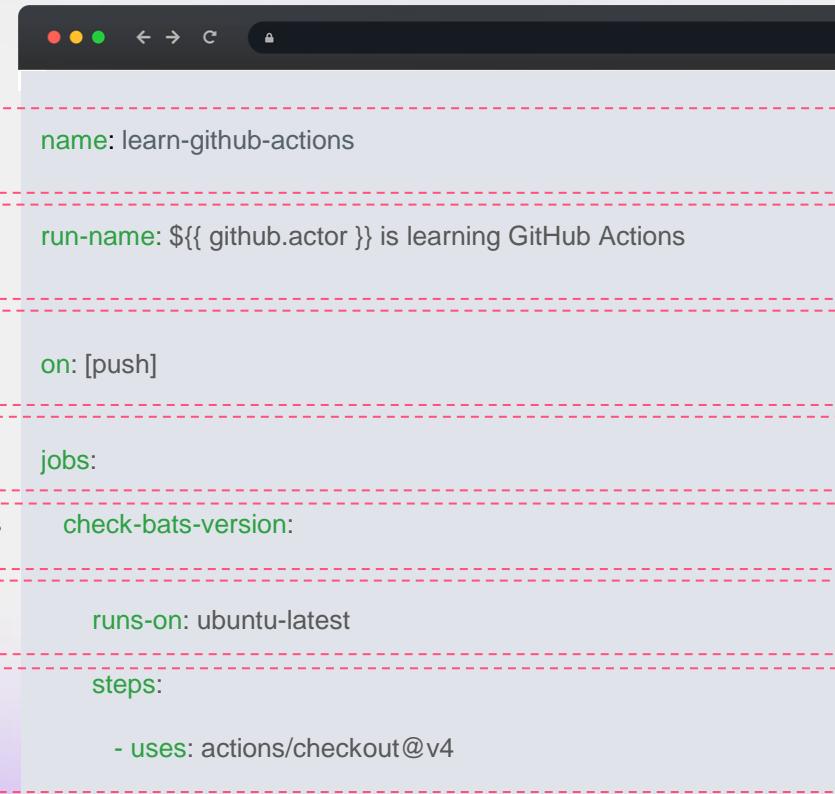
This example uses the push event, so a workflow run is triggered every time someone pushes a change to the repository or merges a pull request

Groups together all the **jobs** that run in the learn-github-actions workflow.

Defines a job named **check-bats-version**. The child keys will define properties of the job.

Configures the **job runner** on the latest version of an Ubuntu Linux runner.

Groups together all the **steps** that run in the **check-bats-version** job and the `uses` keyword specifies that this step will run **v4** of the **actions/checkout** action



```
name: learn-github-actions

run-name: ${{ github.actor }} is learning GitHub Actions

on: [push]

jobs:
  check-bats-version:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
```



Events

Webhook events

- Pull Request
- Push to repo
- Issues
- Releases
-

Scheduled events

Manual events

Events



GitHub

Repository

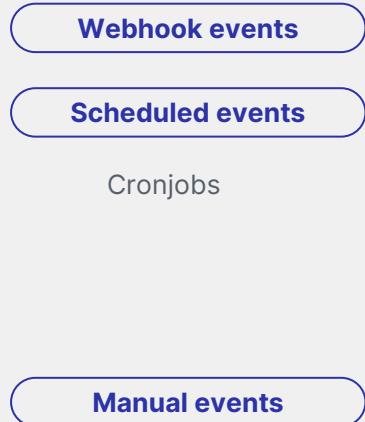
Organization

simple-ci.yaml

```
1 name: Simple
2
3 on:
4   push
5
6 jobs:
7   run_shell_command:
8     name: Actions are amazing
9     runs-on: ubuntu-latest
10    steps:
11      - name: Say hello
12        run: echo "Hello!"
```



Events



The diagram illustrates a GitHub repository structure. On the left, a dashed box labeled "Repository" contains a GitHub logo and the text "simple-ci.yaml". On the right, a solid box labeled "Organization" contains a GitHub logo and the text "simple-ci.yaml". A red arrow points from the "Repository" box to the "on: schedule:" section of the YAML file, which includes a cron expression `30 6 * * 5`.

```
1 name: Simple
2
3 on:
4   schedule:
5     cron: '30 6 * * 5'
6
7 jobs:
8   run_shell_command:
9     name: Actions are amazing
10    runs-on: ubuntu-latest
11    steps:
12      - name: Say hello
13        run: echo "Hello!"
```



Events

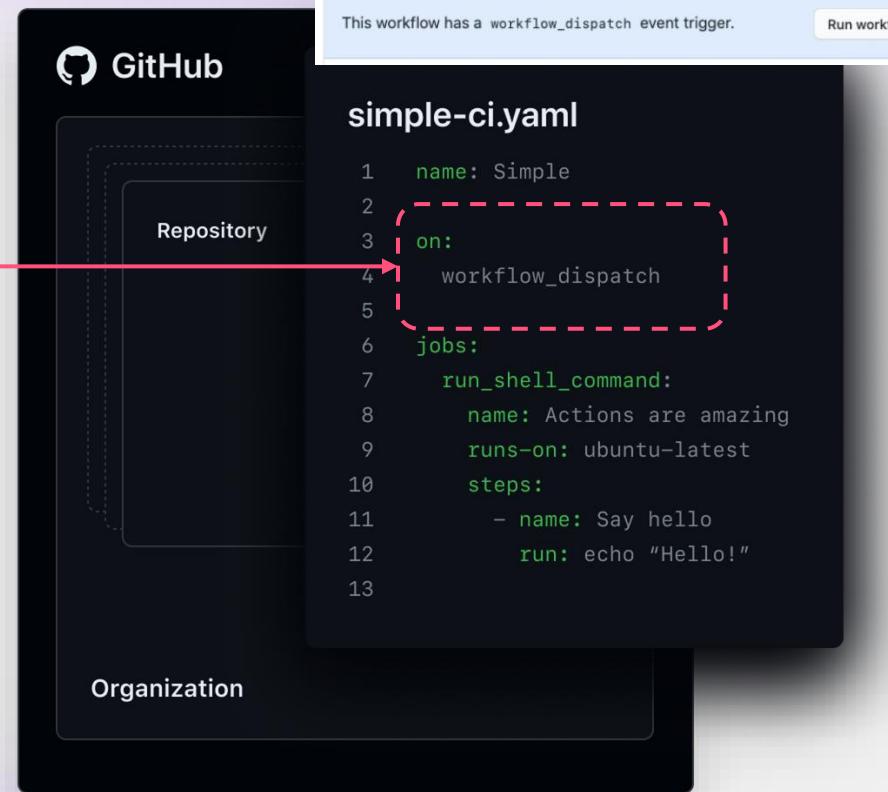
Webhook events

Scheduled events

Manual events

- workflow_dispatch
- repository_dispatch

Events





Runners



GitHub Hosted Runners



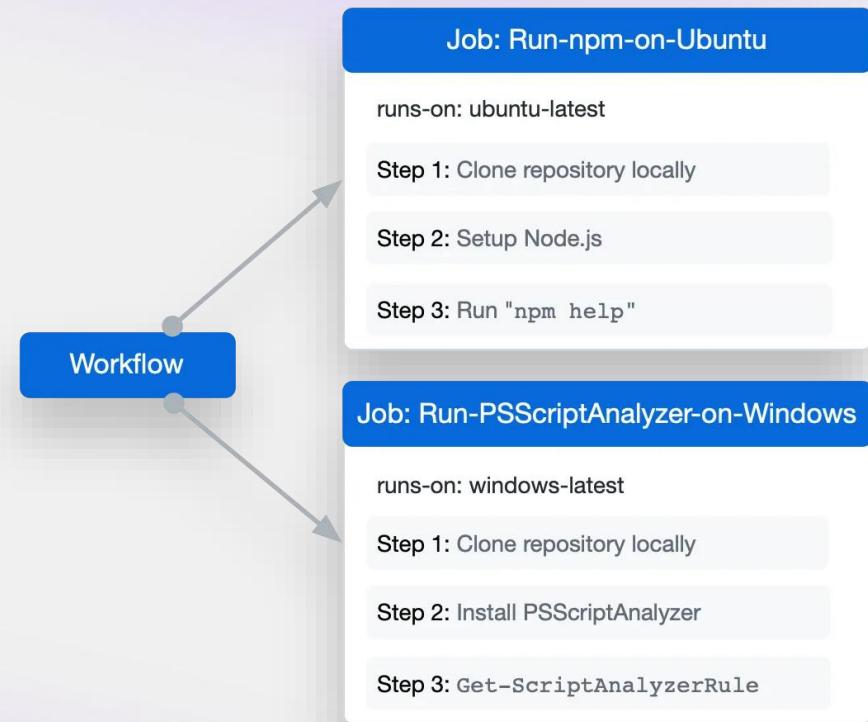
Self-Hosted Runners





GitHub Hosted Runners

- Automatic Provisioning and Maintenance
- Diverse Operating System Support
- Integrated Software and Tools
- Scalability and Workflow Continuity
- Security and Administrative Privileges
- Customization and Local Environment Variables
- Using private vNet on Azure





Self-Hosted Runners

- Full Control and Customization
- Flexibility in Hosting
- Cost Management
- Run on OS not supported on GitHub-hosted runner
- Usage Limits and Continuity
- Custom hardware config
- Can be grouped together

Runner groups

Control access to your runners by specifying the repositories that are able to use your shared organization runners.

New runner group

Group	Runners	⋮
Default ⓘ All repositories, excluding public repositories	100	
azure Selected repositories (18), excluding public repositories	260	⋮
ui-tests Selected repositories (3), excluding public repositories	92	⋮
mobile Selected repositories (1), excluding public repositories	51	⋮
aws Selected repositories (10), excluding public repositories	104	⋮
dotcom Selected repositories (15), excluding public repositories	187	⋮

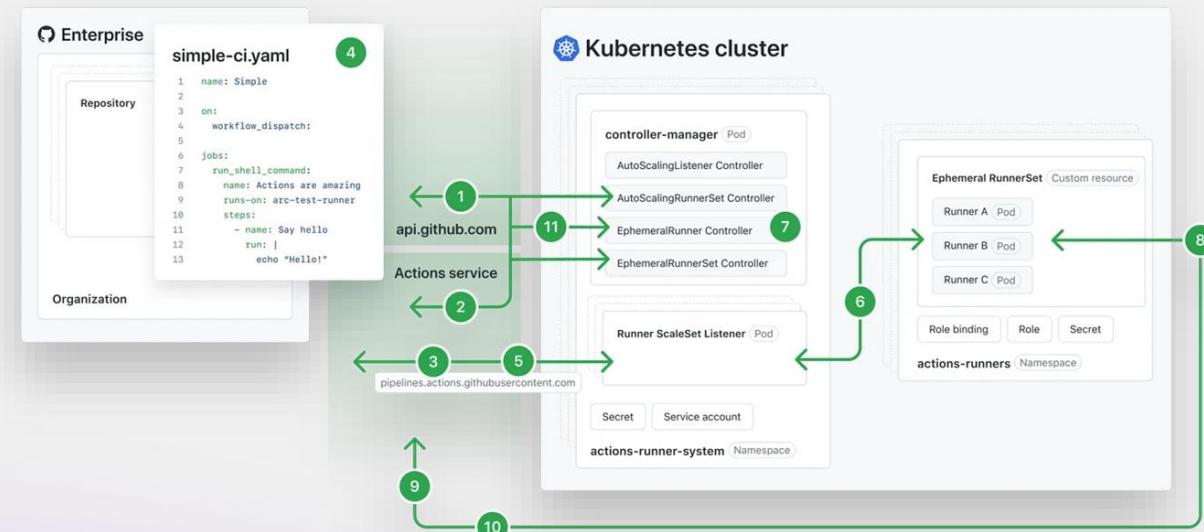
Shared by the Enterprise

Group	Runners	⋮
Default All repositories, excluding public repositories	230	
enterprise-hourly All repositories, excluding public repositories	89	
kubernetes All repositories, excluding public repositories	452	



Self-Hosted Runners

- ✓ Kubernetes Integration
- ✓ Autoscaling Capabilities
- ✓ Ephemeral Runners
- ✓ Container-Based Runners
- ✓ Customizable Installation
- ✓ Runner Container Image





Actions

- Reusable units of code that can be referenced in a workflow
- GitHub runs them in Node.js runtime, or in containers
- Reference an Action, or run scripts directly

The diagram illustrates the scope of GitHub Actions. It shows three nested levels: **Enterprise** (the outermost), **Organization** (the middle), and **Repository** (the innermost). Dashed lines indicate the boundaries between these levels.

```
simple-ci.yaml
1 name: Simple
2
3 on:
4   push
5
6 jobs:
7   run_shell_command:
8     name: Actions are amazing
9     runs-on: ubuntu-latest
10    steps:
11      - name: Say hello
12        run: echo "Hello!"
13      - name: Public Action
14        uses: actions/checkout@v4
15      - name: Local Action
16        uses: ./path/to/action
17      - name: Docker Image
18        uses: docker://alpine:3.8
```

[Script](#)[Public Action](#)[Local Action](#)[Docker Image](#)



Largest Connected Developer Community

“

What I love about GitHub Actions is that you're not limited in your choice of tools.

The screenshot shows a web browser window for GitHub.com displaying the Marketplace. The search bar at the top contains the query "sort:popularity-desc". Below the search bar, there are filters for "Types" (with "Actions" selected), "Actions", and "Stacks". On the left, there's a sidebar with "Categories" including API management, Chat, Code quality, Code review, Continuous integration, Dependency management, Deployment, IDEs, Learning, Localization, and Mobile. On the right, the search results are displayed under the heading "Actions". The results include:

- TruffleHog OSS** By trufflesecurity Scan Github Actions with TruffleHog ★ 9.7k stars
- Super-Linter** By github It is a simple combination of various linters, written in bash, to help validate your source code ★ 8.3k stars
- GitHub Pages action** By peaceiris GitHub Actions for GitHub Pages & Deployments
- yq - parse, merge, and transform YAML** By mikefarah Create, merge, and validate your YAML files ★ 6.8k stars
- Deploy** By James... GitHub Actions for GitHub Pages & Deployments

A large blue callout box in the bottom right corner states "20k+ Actions available in the GitHub Marketplace".



Starter workflows



Preconfigured for specific languages and frameworks



GitHub analyzes your code and suggests the workflows
based on your language and framework



You can also create your own starter workflow to share with
your organization.

The screenshot shows the GitHub interface for choosing a workflow. At the top, there's a navigation bar with the GitHub logo, a search bar containing 'github.com', and a refresh button. Below the header, the title 'Choose a workflow' is displayed, followed by a subtitle: 'Build, test, and deploy your code. Make code reviews, branch management, and issue triaging work the way you want. Select a workflow below.' A link 'Skip this and set up a workflow yourself →' is also present. On the left, a sidebar titled 'Categories' lists 'Automation', 'Continuous integration', 'Deployment', and 'Security' (which is highlighted with a blue background). To the right, a search bar contains the placeholder 'Search workflows'. Below the search bar, it says 'Found 37 workflows'. There are four workflow cards visible:

- CodeQL Analysis** By GitHub: Security analysis from GitHub for C, C++, C#, Go, Java, JavaScript, TypeScript, Python, and Ruby developers. Includes 'Configure' and 'Code scanning' buttons.
- Veracode Static Analysis** By Veracode: Detect fast feedback on flaws with Veracode Static Analysis and the pipeline scanner. Break the build based on flaw severity. Includes 'Configure' and 'Code scanning' buttons.
- Microsoft C++ Code Analysis** By Microsoft: Code Analysis with the Microsoft C & C++ Compiler for CMake based projects. Includes 'Configure' and 'Code scanning' buttons.
- Snyk Infrastructure as Code** By Snyk: Detect vulnerabilities in your infrastructure as code files and surface the issues via GitHub code scanning. Includes 'Configure' and 'Code scanning' buttons.
- Mayhem for API** By ForAllSecure: Automatically test your REST APIs with Mayhem. Includes 'Configure' and 'Code scanning' buttons.
- mobsf** By mobsf: Mobile Security Framework (MobsF) for mobile app security testing. Includes 'Configure' and 'Code scanning' buttons.



Workflow Logs

Build Container
succeeded 12 days ago in 44s

Search logs

Build Container

- > Set up job 5s
- > Checkout 1s
- > Get Jar file artifact 1s
- > Create container image name 0s
- > GitHub Container Registry Login 1s
- > Setup Docker buildx 7s
- > Cache Container layers 2s

Build and Push Container 21s

```
1 ► Run docker/build-push-action@v2
16 ► Docker info
102 /usr/bin/docker buildx build --build-arg VERSION=1.0.0-fix-error-32853982-
SNAPSHOT --build-arg REPOSITORY_NAME=octodemo/bank-books --build-arg
revision=32853982a076c3ad43b6c9de5428132c62902fc --cache-from
type=local,src=/tmp/.buildx-cache --cache-to type=local,dest=/tmp/.buildx-
cache-new --iidfile /tmp/docker-build-push-vtBsI2/iidfile --tag
ghcr.io/octodemo/bank-books:1.0.0-fix-error-32853982-SNAPSHOT --metadata-
file /tmp/docker-build-push-vtBsI2/metadata-file --push .
103 #1 [internal] load build definition from Dockerfile
104 #1 transferring dockerfile: 1.57kB done
105 #1 DONE 0.0s
106
107 #2 [internal] load .dockerrcignore
108 #2 transferring context: 2B done
109 #2 DONE 0.0s
110
```

github.com

octodemo / bank-books Private generated from octodemo/template-bookstore-v2

Code Issues 1 Pull requests 12 Actions Security 339 Insights Settings

fix sql injection warning Build - Test - Publish #18

Summary

Triggered via push 12 days ago pholleran pushed → 3285398 fix-error Status Success Total duration 2m 22s Billable 4m

Jobs

- Build java 11 on ubuntu-20.04
- Build java 11 on windows-latest
- Build Container

Continuous Delivery Deployment

build_test_publish.yml on: push

Matrix: build

```
graph LR; A[Build java 11 on ubuntu-20.04 16s] --> B[Build Container 44s]; C[Build java 11 on windows-latest 1m 11s]
```

Artifacts

Produced during runtime

Name	Size
standalone-ubuntu-20.04-11.jar	13.7 MB
standalone-windows-latest-11.jar	13.7 MB



Advanced Syntax

permissions

Set workflow permissions for GITHUB_TOKEN

env

Set environment variables for all run steps

defaults

Set the shell and working directory for the run

concurrency

Manage workflows running concurrently

format

Format outputs

needs

Make jobs dependent of each other. Share outputs

```
if success() always() cancelled()  
failure()  
Check whether a job should run based on  
variables.
```

timeout-minutes

Limit runtime

continue-on-error

Handle termination of workflows

join

Join arrays into strings

services

Create sidecar docker images for integration dependencies

container

Use a container for the steps execution

contains

Check if a string is contained in another

startsWith/endsWith

Check start/end of a string

toJSON/fromJSON

Make string JSON and JSON strings



Usage Limits

Limit	Description	Notes
Concurrent Jobs <small>(Based on Enterprise Plan) This is for the entire enterprise</small>	GHEC 500 32 core: 1,000 concurrent jobs 64 core: 1,000 concurrent jobs	50 maximum concurrent macOS jobs
Job Execution	6 hours	On error, jobs terminated and seen as failed. This limit doesn't apply to self-hosted runners
Workflow Run	35 days	On error, workflow cancelled when reached. This limit also applies to self-hosted runners
Job Queue	24 hours	Jobs terminated when the limit is reached. This limit only applies to self-hosted runners
API Requests	15,000 per hour per repo	On error, API calls fail. Can cause jobs to fail. This limit also applies to self-hosted runners
Job Matrix	256 per run	This limit also applies to self-hosted runners



Actions Environments and secrets





Environments

Environments are used to describe a general deployment target like **production**, **staging**, or **development**.



Dynamic Creation

- Can be dynamically created based on the needs, allowing for flexible and scalable deployment targets.
- Automated setup and teardown of environments, ensuring efficient resource management

Review & Approval

- Support a review and approval process
- Allows for designated reviewers to approve or reject deployments

Protection Rules

- Protection rules can be applied to environments, restricting access and enforcing policies
- Can include requirements such as specific branch protections, status checks



Environments manage and secure deployment targets like production

Control deployments

Add gated deployments with approvals

Control secrets and envs variables

Review all deployments

Navigate directly to urls for deployments

Fully integrated with the checks API

Supports matrix for gated deployments

github.com

Environments / Configure Production

Deployment protection rules

Configure reviewers, timers, and custom rules that must pass before deployments to this environment can proceed.

Required reviewers
Specify people or teams that may approve workflow runs when they access this environment.

Add up to 6 more reviewers
Search for people or teams...

Prevent self-review
Require a different approver than the user who triggered the workflow run.

Wait timer
Set an amount of time to wait before allowing deployments to proceed.

Enable custom rules with GitHub Apps Beta
[Learn about existing apps](#) or [create your own protection rules](#) so you can deploy with confidence.

Allow administrators to bypass configured protection rules

Save protection rules

Deployment branches and tags

Limit which branches and tags can deploy to this environment based on rules or naming patterns.

No restriction ▾



Secrets

GitHub Actions Secrets securely store sensitive data, manage access, and ensure encryption



Storage & Access

- Allow you to store sensitive information securely at the repo, environment, and org levels
- Access to secrets can be controlled through policies

Encryption & Security

- Secrets are encrypted before storage & during transmission and storage
- Redacts secrets from logs and allows to use short-lived tokens

Integration & Management

- Secrets can be integrated into workflows by setting them as inputs or environment variables
- Management of secrets is facilitated through the GitHub REST API and GitHub CLI



secrets ensure secure handling of credentials and sensitive data across various scenarios

Built-in secret store

Encrypted (LibSodium sealed box)

Use directly from your workflow

Redacted in workflow logs

API & CLI support

Organization / repository / environment secrets

The screenshot shows the GitHub 'Secrets' page with the following sections:

- General**: A sidebar with links to Access, Collaborators and teams, Team and member roles, Code and automation, Branches, Tags, Rules, Actions, Webhooks, Copilot, Environments, Pages, and Custom properties.
- Actions secrets and variables**: A main content area with a heading 'Actions secrets and variables'. It explains that secrets and variables allow managing reusable configuration data. Secrets are encrypted and used for sensitive data, while variables are plain text for non-sensitive data.
- Environment secrets**: A table listing environment secrets. One entry is shown: MY_ENV_SECRET (Production, last updated 4 minutes ago).
- Repository secrets**: A table listing repository secrets. One entry is shown: MY_REPO_SECRET (last updated 4 minutes ago, with edit and delete icons).
- Organization secrets**: A table listing organization secrets. One entry is shown: MY_ORG_SECRET (last updated 4 minutes ago).

The 'Actions' link in the sidebar is highlighted with a blue bar.



Environment secrets

- ✓ Built-in secret store

- ✓ Encrypted (LibSodium sealed box)

Repository secrets

- ✓ Use directly from your workflow

- ✓ Redacted in workflow logs

Organization secrets

- ✓ API & CLI support

- ✓ Organization / repository / environment secrets

The screenshot shows the GitHub 'Secrets' settings page with three main sections: Environment secrets, Repository secrets, and Organization secrets. Each section has a 'Manage [section] secrets' button.

- Environment secrets:** Contains a single secret named 'MY_ENV_SECRET' in the Production environment, last updated 4 minutes ago.
- Repository secrets:** Contains a single secret named 'MY_REPO_SECRET', last updated 4 minutes ago. It includes edit and delete icons.
- Organization secrets:** Contains a single secret named 'MY_ORG_SECRET', last updated 4 minutes ago.

The sidebar on the left lists various GitHub features: General, Access, Collaborators and teams, Team and member roles, Code and automation, Branches, Tags, Rules, Actions, Webhooks, Copilot, Environments, Pages, Custom properties, Security, Code security and analysis, Deploy keys, Secrets and variables (which is the active tab), Actions, Codespaces, and Dependabot.



Using secrets in workflows



All secrets can be accessed using the same syntax;

- `${{ secrets.<SECRET_NAME> }}`



Every workflow run provisions a `GITHUB_TOKEN` secret by default

- Scoped to a single repository
- Enterprise/organization/repository policies for default permissions
- `permissions` syntax for granular permissions on workflow- or job-level
- Can't trigger other workflows



Marketplace Actions exist for integration with other secret stores

```
sample-workflow.yml
name: Pull request labeler
on:
  pull_request:
jobs:
  triage:
    runs-on: ubuntu-latest
    permissions:
      contents: read
      actions: read
      issues: write
    steps:
      - uses: actions/labeled@v2
        with:
          repo-token: ${{ secrets.GITHUB_TOKEN }}
      - uses: myaction@v1
        with:
          mySecret: ${{ secrets.MY_SECRET }}
```



Vault Secrets

By hashicorp

A Github Action that allows you to consume HashiCorp Vault™ secrets as secure environment variables



Azure key vault - Get Secrets

By Azure

Get Secrets from Azure Key Vault instance and set as output variables.
github.com/azure/actions



Permissions for GITHUB_TOKEN

Token Creation: Automatically created for each workflow job

Token Scope: Limited to the repository containing the workflow

Modify Permission: Adjustable in workflow files for specific needs

Default Permissions: Admins can set to permissive or restricted

Additional Permissions: Use GitHub App or personal access tokens

```
sample-workflow.yml > name
1   name: Pull request labeler
2
3   on:
4     pull_request:
5
6   jobs:
7     triage:
8       runs-on: ubuntu-latest
9
10  permissions:
11    actions: read|write|none
12    checks: read|write|none
13    contents: read|write|none
14    deployments: read|write|none
15    issues: read|write|none
16    packages: read|write|none
17    pull-requests: read|write|none
18    repository-projects: read|write|none
19    security-events: read|write|none
20    statuses: read|write|none
21
22  steps:
23    - uses: actions/labeled@v2
24
25 Workflow permissions
26 Choose the default permissions granted to the GITHUB_TOKEN when running workflows in this organization. You can specify more granular
27 permissions in the workflow using YAML. Learn more about managing permissions.
28 Repository administrators will only be able to change the default permissions to a more restrictive setting.
29
30  Read and write permissions
31   Workflows have read and write permissions in the repository for all scopes.
32  Read repository contents and packages permissions
33   Workflows have read permissions in the repository for the contents and packages scopes only.
34
35 Choose whether GitHub Actions can create pull requests or submit approving pull request reviews.
36  Allow GitHub Actions to create and approve pull requests
37
38 Save
```



Manage Actions

Managing workflows & Actions



Actions policies



Configure Policies: Apply at enterprise, organization, or repository levels



Restrict Actions: Limit which GitHub Actions can be used



Artifact Retention: Set retention periods for artifacts and logs



Fork PR Workflows: Control workflows from forked pull requests



GITHUB_TOKEN Permissions: Define default token permissions



Audit Logs: Track and review actions usage and modifications

The screenshot shows the "General actions permissions" section of the GitHub Actions policies configuration. It includes sections for "Policies", "Runners", "Artifact and log retention", and "Fork pull request workflows from outside collaborators".

Policies: Choose which repositories are permitted to use GitHub Actions. The dropdown is set to "All repositories".
Options:

- Allow all actions and reusable workflows**: Any action or reusable workflow can be used, regardless of who authored it or where it is defined.
- Allow enterprise actions and reusable workflows**: Any action or reusable workflow defined in a repository within the enterprise can be used.
- Allow enterprise, and select non-enterprise, actions and reusable workflows**: Any action or reusable workflow that matches the specified criteria, plus those defined in a repository within the enterprise, can be used. [Learn more about allowing specific actions and reusable workflows to run](#).

Runners: Choose which repositories are allowed to create repository-level self-hosted runners. The dropdown is set to "All repositories".

Artifact and log retention: Choose the default repository settings for artifacts and logs. The dropdown is set to "90 days".
Your enterprise has set a maximum limit of 90 days. [Learn more about the artifact and log retention policy](#).

Fork pull request workflows from outside collaborators: Choose which subset of outside collaborators will require approval to run workflows on their pull requests. The dropdown is set to "Require approval for first-time contributors who are new to GitHub".
Options:

- Require approval for first-time contributors who are new to GitHub**: Only first-time contributors who recently created a GitHub account will require approval to run workflows.
- Require approval for first-time contributors**: Only first-time contributors will require approval to run workflows.
- Require approval for all outside collaborators**



Sharing Workflows



Reusing Workflows

One workflow can be called from another, allowing to reuse workflows



Starter Workflows

Organizations can create starter workflows that act as templates



Centralized Secrets and Variables

Manage secrets and variables at the organization level



Self-Hosted Runners

Organizations can group runners and control repository access via policies



Centralized Management

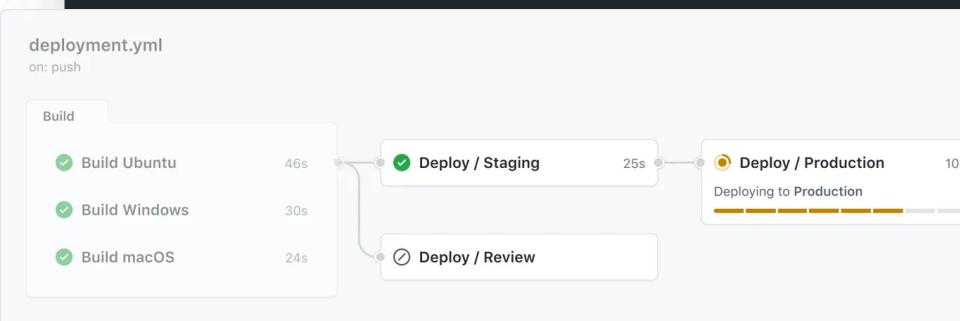
Use organizational settings to centrally manage workflows, secrets, and runners, enhancing collaboration and security.



Reusing workflows

- ✓ Composite Actions: bundle multiple steps into a single action, enabling reuse across different workflows
- ✓ Use the `workflow_call` trigger to call reusable workflows from other workflows within the same organization
- ✓ Design workflows to be modular and reusable by encapsulating common tasks and processes
- ✓ Use inputs and outputs to parameterize reusable workflows
- ✓ Maintain reusable workflows in a central repository

```
name: Reusable workflow example
on:
  workflow_call:
    inputs:
      config-path:
        required: true
        type: string
    secrets:
      token:
        required: true
jobs:
  triage:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/lableler@v4
        with:
          repo-token: ${{ secrets.token }}
          configuration-path: ${{ inputs.config-path }}
```





Caching in GitHub Workflows



Use the `actions/cache` action to create and restore caches, identified by unique keys



Cache keys can be customized using contexts and expressions. The action supports `restore-keys` for fallback options in case of a cache miss

[Caching dependencies to speed up workflows](#)

Caching can help with speeding up workflows when you need to install dependencies. NPM, Python, Ruby, etc... these are simple examples of applications that require dependencies to be built. But there are more complex scenarios, such as Java, C/C++ and modularized microservices that often require downstream artifacts. Caching can speed up your builds when your dependencies have not changed



Caches not accessed in over 7 days are removed, and repositories have a 10 GB cache storage limit



View, filter, sort, and delete cache entries via the GitHub web interface or REST API, allowing better control over cache usage



Best Practices on Actions in an organization



Implementing these best practices can help organizations effectively manage their CI/CD pipelines using GitHub Actions, enhancing security, efficiency, and collaboration.

Security

- Limit Permissions
- Pin Actions to SHAs
- Use Encrypted Secrets

Efficiency

- Caching
- Concurrency and Timeouts

Reusability

- Reusable Workflows
- Leverage Starter Workflows

Runners

- Controlled Usage
- Security Measures

Compliance

- Define Clear Rules
- Innersource Practices



Building Actions

Build Actions &

Workflows



Write Custom Action



Actions Types



JavaScript Actions



Docker Actions



Composite Actions



Repository Structure



action.yml



Dockerfile



Source code files

```
name: "Hello Action"
description: "Greet someone"
author: "octocat@github.com"

inputs:
  my_name:
    description: "Who to greet"
    required: true
    default: "World"

outputs:
  greeting:
    description: "Full greeting"

runs:
  using: "docker"
  image: "Dockerfile"

branding:
  icon: "mic"
  color: "purple"
```



Composite Action

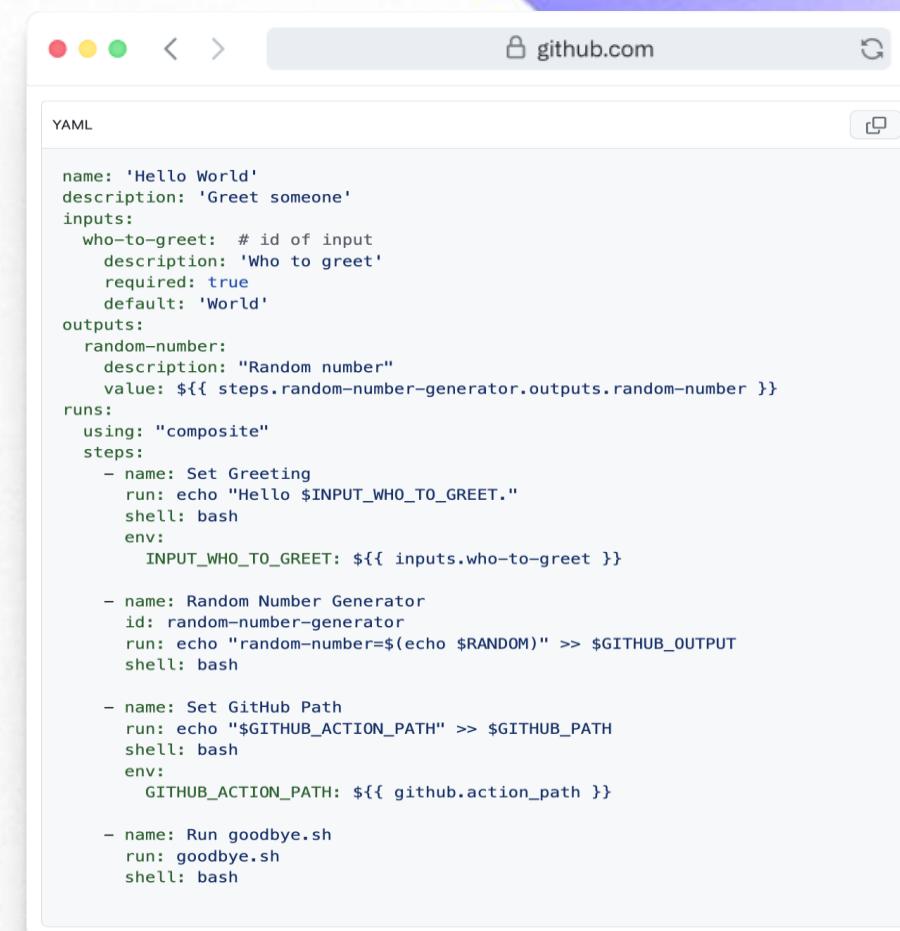
A Composite Action in GitHub is an action that combines multiple individual steps into one reusable component. Instead, they use YAML syntax to define a sequence of steps that are executed together as a single unit. This allows you to encapsulate common tasks and reuse them across different workflows, improving maintainability and reducing redundancy.

action.yml

Metadata File

Steps

A series of steps defined in the `action.yml`



The screenshot shows a GitHub browser window with the URL `github.com`. The page displays a YAML configuration file for a composite action named "Hello World". The YAML code defines inputs ("who-to-greet"), outputs ("random-number"), runs ("using: composite" with steps), and env variables. The steps include setting a greeting, generating a random number, setting the GitHub path, and running a goodbye script.

```
name: 'Hello World'
description: 'Greet someone'
inputs:
  who-to-greet: # id of input
    description: 'Who to greet'
    required: true
    default: 'World'
outputs:
  random-number:
    description: "Random number"
    value: ${{ steps.random-number-generator.outputs.random-number }}
runs:
  using: "composite"
  steps:
    - name: Set Greeting
      run: echo "Hello $INPUT_WHO_TO_GREET."
      shell: bash
      env:
        INPUT_WHO_TO_GREET: ${{ inputs.who-to-greet }}

    - name: Random Number Generator
      id: random-number-generator
      run: echo "random-number=$(echo $RANDOM)" >> $GITHUB_OUTPUT
      shell: bash

    - name: Set GitHub Path
      run: echo "$GITHUB_ACTION_PATH" >> $GITHUB_PATH
      shell: bash
      env:
        GITHUB_ACTION_PATH: ${{ github.action_path }}

    - name: Run goodbye.sh
      run: goodbye.sh
      shell: bash
```



JavaScript Action

A JavaScript action is a custom action written in JavaScript or TypeScript that runs directly on the GitHub-hosted runner or a self-hosted runner. It allows you to leverage the full power of Node.js along with GitHub's rich API and automation capabilities. JavaScript actions are ideal for tasks that require high performance and need to interact directly with the GitHub environment.

action.yml

Metadata File

index.js

Action Code

package.json

Package Configuration

The screenshot shows a GitHub browser window with the URL `github.com`. The page displays two files: `action.yml` and `index.js`.

YAML

```
name: 'Hello World'
description: 'Greet someone and record the time'
inputs:
  who-to-greet: # id of input
    description: 'Who to greet'
    required: true
    default: 'World'
outputs:
  time: # id of output
    description: 'The time we greeted you'
runs:
  using: 'node20'
  main: 'index.js'
```

JavaScript

```
const core = require('@actions/core');
const github = require('@actions/github');

try {
  // `who-to-greet` input defined in action metadata file
  const nameToGreet = core.getInput('who-to-greet');
  console.log(`Hello ${nameToGreet}`);
  const time = (new Date()).toTimeString();
  core.setOutput("time", time);
  // Get the JSON webhook payload for the event that triggered the workflow
  const payload = JSON.stringify(github.context.payload, undefined, 2)
  console.log(`The event payload: ${payload}`);
} catch (error) {
  core.setFailed(error.message);
}
```



Docker Action

A Docker Action is a custom GitHub Action that runs inside a Docker container. This type of action leverages the capabilities of Docker to create a portable and reproducible environment. Docker Actions are ideal for scenarios where you need a specific software environment, have complex dependencies, or require a high level of isolation.

Dockerfile

Dockerfile for environments

action.yml

Action Metadata File

entrypoint.js

Entrypoint Script

The screenshot shows a GitHub browser window with the URL github.com. The page displays the configuration for a GitHub Action named 'Hello World'. It includes a YAML file and a Dockerfile.

YAML

```
# action.yml
name: 'Hello World'
description: 'Greet someone and record the time'
inputs:
  who-to-greet: # id of input
    description: 'Who to greet'
    required: true
    default: 'World'
outputs:
  time: # id of output
    description: 'The time we greeted you'
runs:
  using: 'docker'
  image: 'Dockerfile'
  args:
    - ${{ inputs.who-to-greet }}
```

Dockerfile

```
# Container image that runs your code
FROM alpine:3.10

# Copies your code file from your action repository to the filesystem path `/` of
# the container
COPY entrypoint.sh /entrypoint.sh

# Code file to execute when the docker container starts up (`entrypoint.sh`)
ENTRYPOINT ["/entrypoint.sh"]
```



Best Practices

Write your own Action



Creating your own GitHub Actions can significantly enhance the automation of your CI/CD pipelines and improve your workflow efficiency.

Security

- Limit Permissions
- Pin Dependencies
- Handle Secrets Securely

Documentation

- README File
- Provide Examples

Testing

- Local Testing
- Validation

Reusability

- Reusable Components
- Composite Actions
- Use GitHub Actions Toolkit

Optimization

- Efficient Code
- Caching



Migration

Migrate to Actions



Faster onboarding

**Migrating to GitHub Actions
can streamline workflows,
improve efficiency, and
leverage the extensive
GitHub ecosystem.**



GitHub Actions Importer

The [GitHub Actions Importer](#) Tool is a powerful utility designed to facilitate the migration of existing CI/CD pipelines from other platforms to GitHub Actions.

```
gh actions-importer audit -h
```

```
gh actions-importer forecast -h
```

```
gh actions-importer dry-run -h
```

```
gh actions-importer migrate -h
```

The screenshot shows a macOS terminal window with the URL github.com in the address bar. The terminal has three tabs open: Bash, Python, and GitHub Actions Importer. The GitHub Actions Importer tab contains the following content:

- 1 Install the GitHub Actions Importer CLI extension:
Bash
gh extension install github/gh-actions-importer
- 2 Verify that the extension is installed:
\$ gh actions-importer -h
Options:
-?, -h, --help Show help and usage information

Commands:
update Update to the latest version of GitHub Actions Importer.
version Display the version of GitHub Actions Importer.
configure Start an interactive prompt to configure credentials used to authenticate with your CI server(s).
audit Plan your CI/CD migration by analyzing your current CI/CD footprint.
forecast Forecast GitHub Actions usage from historical pipeline utilization.
dry-run Convert a pipeline to a GitHub Actions workflow and output its yaml file.
migrate Convert a pipeline to a GitHub Actions workflow and open a pull request with the changes.



GitHub Actions Importer

The [GitHub Actions Importer](#) supports migration from various popular CI/CD platforms, enabling organizations to seamlessly transition their existing pipelines to GitHub Actions.

Jenkins

Azure

Travis CI

Bitbucket

CircleCI

Bamboo

GitLab CI

TeamCity

```
$ gh actions-importer migrate -h
Description:
  Convert a pipeline to a GitHub Actions workflow and open a pull request with the changes.

[...]

Commands:
  azure-devops  Convert an Azure DevOps pipeline to a GitHub Actions workflow and open a pull request with the changes.
  bamboo        Convert a Bamboo pipeline to GitHub Actions workflows and open a pull request with the changes.
  circle-ci     Convert a CircleCI pipeline to GitHub Actions workflows and open a pull request with the changes.
  gitlab        Convert a GitLab pipeline to a GitHub Actions workflow and open a pull request with the changes.
  jenkins       Convert a Jenkins job to a GitHub Actions workflow and open a pull request with the changes.
  travis-ci      Convert a Travis CI pipeline to a GitHub Actions workflow and open a pull request with the changes.
```



Runners

GitHub Actions Runners



GitHub Runners

GitHub Hosted Runners

- **Managed Environment**
 - Maintenance
 - Pre-Installed Software
- **Resource Allocation**
 - Scalability
 - Cost
- **Security**
 - Isolation
 - Network Access
- **Ease of Use**
 - Quick Setup
 - Limited Customization
- **Performance**
 - Standard Performance

Self-hosted Runners

- **Custom Environment**
 - Full Control
 - Customization
- **Resource Allocation**
 - Scalability
 - Cost
- **Security**
 - Responsibility
 - Network Access
- **Ease of Use**
 - Setup Complexity
 - Flexibility
- **Performance**
 - Custom Performance



Self-Hosted Runners

[Self-hosted runners](#) are machines that you manage and maintain to execute jobs from GitHub Actions workflows.

Provision the Runner

Install the Runner Application

Configure the Runner

Manage and Maintain

The screenshot shows the GitHub interface for managing self-hosted runners. At the top, there's a header with the GitHub logo and the URL "github.com". Below the header, the main content area has a title "Add new self-hosted runner" and a note about agreeing to GitHub's Terms of Service or Corporate Terms of Service. It includes sections for "Runner Image" (selected "macOS"), "Architecture" (selected "x64"), and "Download" which contains a command-line script for provisioning the runner. Below this is a "Configure" section with a command-line script for starting the configuration experience. Further down is a "Using your self-hosted runner" section with a YAML snippet for a workflow file. At the bottom, there's a "Runners" management page with tabs for "Policies", "Runners" (selected), and "Runner groups". It shows a search bar and a table of runners. A "New runner" button is at the top right of this section. To the right, there are two boxes: one for "New GitHub-hosted runner" (described as pay-as-you-go, customizable, secure, scaled & managed by GitHub) and another for "New self-hosted runner" (described as bringing your own infrastructure).

Add new self-hosted runner

Adding a self-hosted runner requires that you download, configure, and execute the GitHub Actions Runner. By downloading and configuring the GitHub Actions Runner, you agree to the [GitHub Terms of Service](#) or [GitHub Corporate Terms of Service](#), as applicable.

Runner Image

macOS Linux Windows

Architecture

x64

Download

```
# Create a folder  
$ mkdir actions-runner && cd actions-runner  
# Download the latest runner package  
$ curl -o actions-runner-osx-x64-2.317.0.tar.gz -L https://github.com/actions/runner/releases/download/v2.317.0/actions-runner-osx-x64-2.317.0.tar.gz  
# Optional: Validate the hash  
$ echo "0b23ee79731522d9e1229d14d6c200e06ac9d7dddf5641966209a7708a43c14" actions-runner-osx-x64-2.317.0.tar.gz | shasum -a 256 -c  
# Extract the installer  
$ tar xzf ./actions-runner-osx-x64-2.317.0.tar.gz
```

Configure

```
# Create the runner and start the configuration experience  
$ ./config.sh --url https://github.com/enterprises/mousismail --token AW6AFIFVLUYYJ6H275LQUELGMCGE  
# Last step, run it!  
$ ./run.sh
```

Using your self-hosted runner

```
# Use this YAML in your workflow file for each job  
runs-on: self-hosted
```

For additional details about configuring, running, or shutting down the runner, please check out our [product docs](#).

Policies **Runners** Runner groups

Includes all runners across self-hosted and GitHub-hosted runners.

Search runners

Runners

Standard GitHub-hosted runners Pay-as-you-go, customizable, secure, scaled & managed by GitHub

ubuntu-latest-m ubuntu-latest-m Runner group: Default Larger Runners Public IP: Disabled

New GitHub-hosted runner

New self-hosted runner



Actions Runner Controller

ARC (Action Runner Controller) is an open-source project that integrates GitHub Actions with Kubernetes, enabling the automatic management of self-hosted runners.

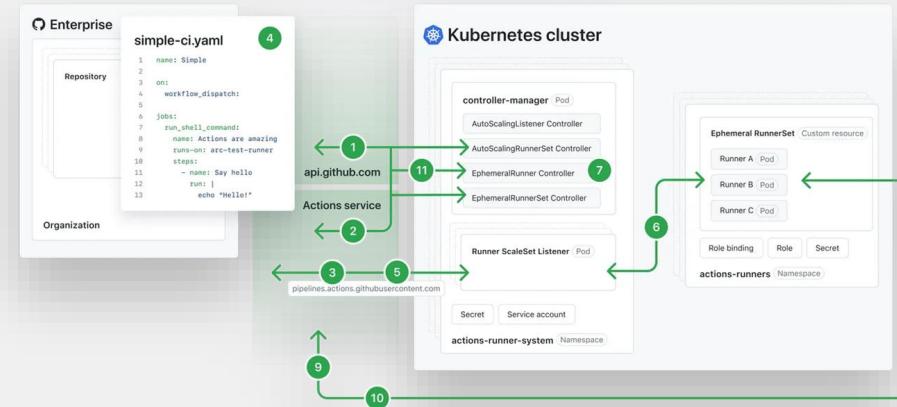
Runner Deployment

Auto-Scaling

Custom Resource Definitions

Security and Isolation

Monitoring and Logging





Security With Self-hosted Runners



Using self-hosted runners for GitHub Actions offers greater flexibility and control over the environment, but it also introduces additional security considerations.

Runner Isolation

- Isolation of Runners
- Ephemeral Runners

Network Security

- Restricted Network Access
- VPN and VPC

Access Control

- Least Privilege Principle
- Access Tokens

Environment

- Secure Configuration
- Monitoring and Logging

Data

- Secrets Management
- Encryption



Best Practices With Self-hosted Runners



Securing self-hosted runners for GitHub Actions requires careful planning and adherence to best practices

Runner Lifecycle Management

- Automate Provisioning
- Automate Updates

Containerization

- Use Containers
- Custom Container Images

Ephemeral Runners

- Short-Lived Runners
- Scale on Demand

Audits

- Conduct Audits
- Compliance Checks



CI/CD

Action as CI/CD workflows



Basic CI/CD Action

GitHub Actions is a powerful feature within GitHub that enables the automation of workflows, particularly for continuous integration (CI) and continuous deployment (CD)



Build & Test

This guide shows you how to build, test, and publish a Go package

Specifying a Go version

Installing dependencies

Caching dependencies

Building and testing your code

```
YAML
name: Upload Go test results

on: [push]

jobs:
  build:

    runs-on: ubuntu-latest
    strategy:
      matrix:
        go-version: [ '1.19', '1.20', '1.21.x' ]

    steps:
      - uses: actions/checkout@v4
      - name: Setup Go
        uses: actions/setup-go@v5
        with:
          go-version: ${{ matrix.go-version }}
      - name: Install dependencies
        run: go get .
      - name: Test with Go
        run: go test -json > TestResults-${{ matrix.go-version }}.json
      - name: Upload Go test results
        uses: actions/upload-artifact@v4
        with:
          name: Go-results-${{ matrix.go-version }}
          path: TestResults-${{ matrix.go-version }}.json
```



Deploy

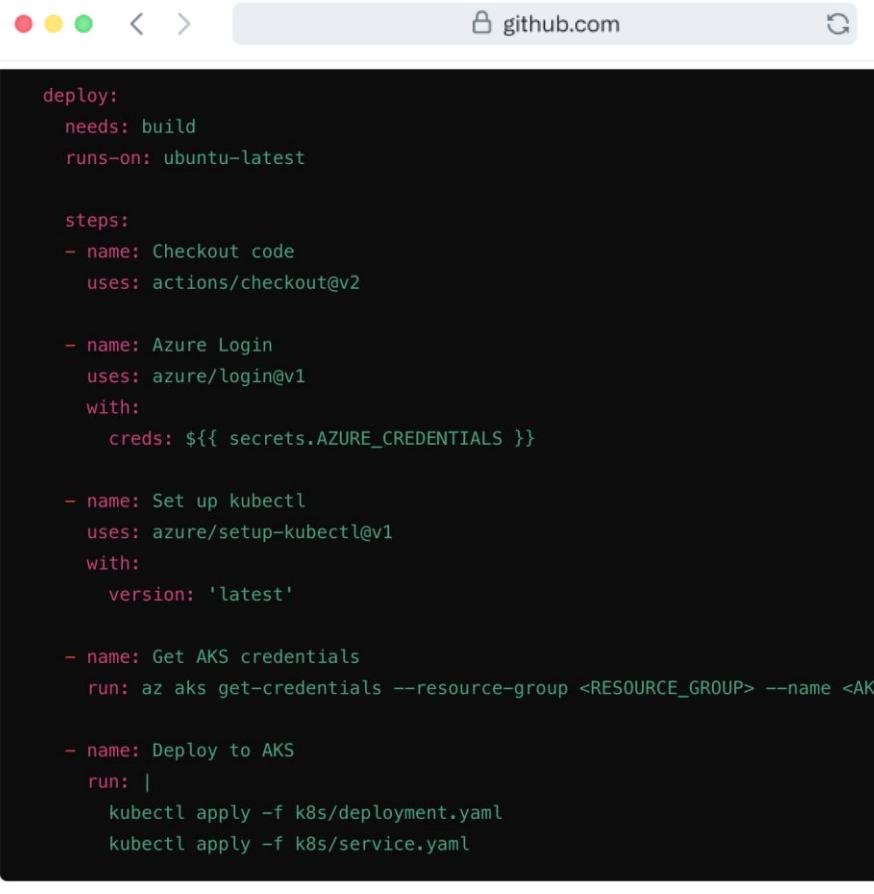
This guide explains how to use GitHub Actions to build and deploy a project to Azure Kubernetes Service

Azure Login

Build image on ACR

Configure deployment

Deploys application



The screenshot shows a GitHub Actions workflow file in a browser window. The URL is github.com. The workflow is titled "Deploy" and contains the following YAML code:

```
deploy:
  needs: build
  runs-on: ubuntu-latest

  steps:
    - name: Checkout code
      uses: actions/checkout@v2

    - name: Azure Login
      uses: azure/login@v1
      with:
        creds: ${{ secrets.AZURE_CREDENTIALS }}

    - name: Set up kubectl
      uses: azure/setup-kubectl@v1
      with:
        version: 'latest'

    - name: Get AKS credentials
      run: az aks get-credentials --resource-group <RESOURCE_GROUP> --name <AKS_NAME>

    - name: Deploy to AKS
      run:
        kubectl apply -f k8s/deployment.yaml
        kubectl apply -f k8s/service.yaml
```



Thanks