

# 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

04

### Demos

Actions Demos



# 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



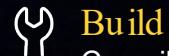
## Plan

Triage 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



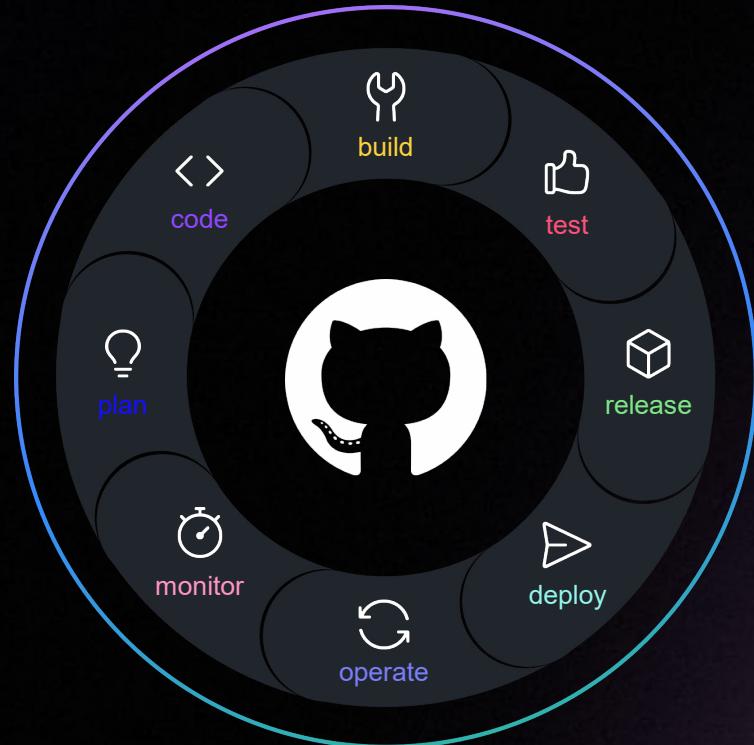
## Release

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

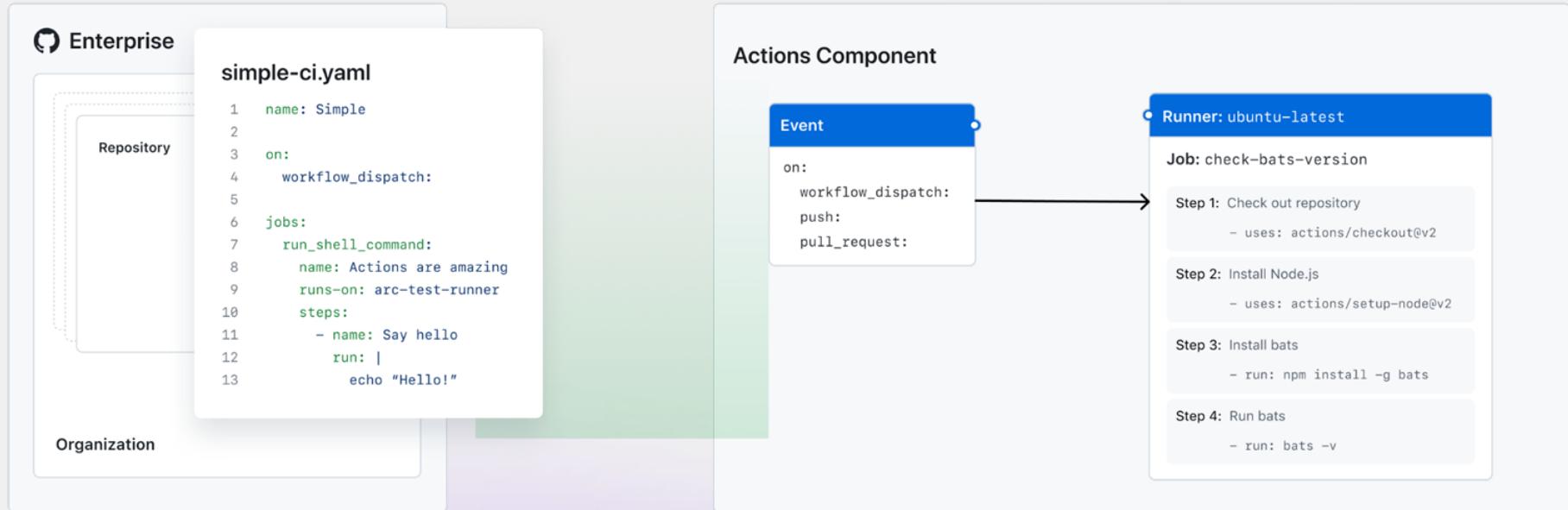
The screenshot shows the GitHub Actions interface. At the top, there's a navigation bar with links for Discussions, Actions (which is active), Projects, Wiki, Security, Insights, and Settings. Below the navigation, a summary table shows a single workflow run triggered manually 2 minutes ago by octodemobot, with a status of In progress, total duration of -, and no artifacts.

The main area displays a workflow named "create\_demo.yml" with one job, "on: workflow\_dispatch". The job has three steps: "Initialize" (status: success), "Configure Creation Steps" (status: success), and "Create Demo Repository" (status: in progress). A tooltip for the third step indicates it is "Deploying to octodemo/testing-provisioning". To the right of the workflow, there are several options: "Register Demo Deployment S...", "Update tracking issue", "Matrix: Configure secrets", and "Waiting for pending jobs".

- ✓ 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



## simple-ci.yaml

```
1  name: Simple
2
3  on:
4    workflow_dispatch:
5
6  jobs:
7    run_shell_command:
8      name: Actions are amazing
9      runs-on: arc-test-runner
10     steps:
11       - name: Say hello
12         run: |
13           echo "Hello!"
```

## Actions Component

### Event

```
on:
  workflow_dispatch:
  push:
  pull_request:
```

### Runner: ubuntu-latest

```
Job: check-bats-version
Step 1: Check out repository
  - uses: actions/checkout@v2

Step 2: Install Node.js
  - uses: actions/setup-node@v2

Step 3: Install bats
  - run: npm install -g bats

Step 4: Run bats
  - run: bats -v
```

Organization

Repository

Enterprise



# 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**.

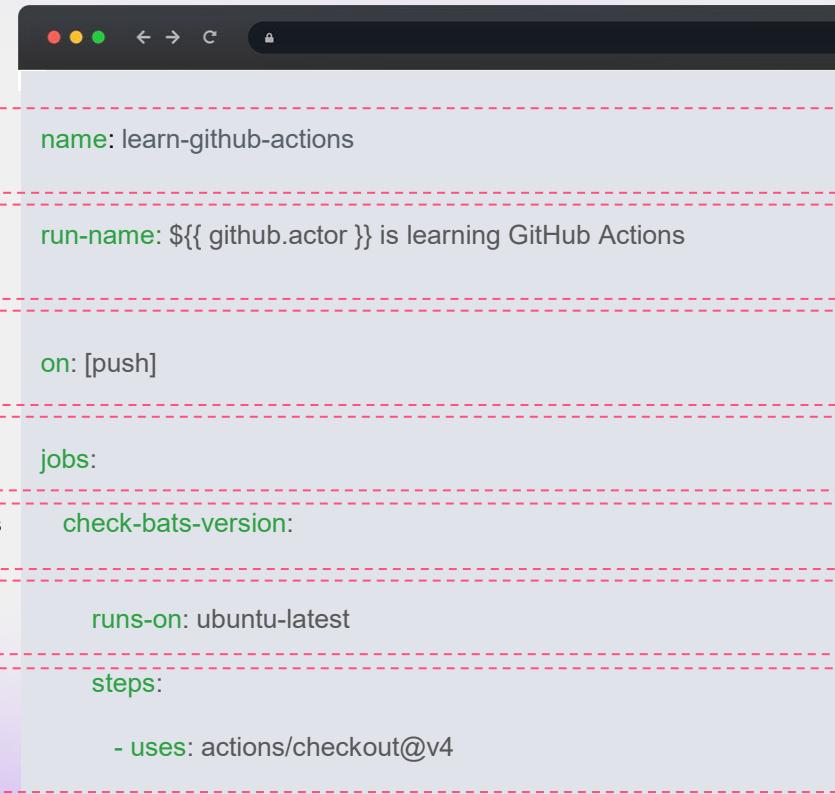
This example uses the push event, so a workflow run is triggered every time someone pushes a change to the repository or merges a pull request

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

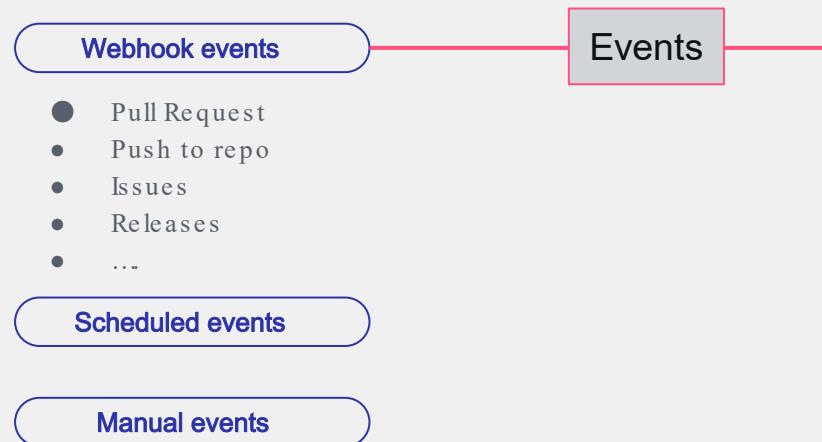
run-name: ${{ github.actor }} is learning GitHub Actions

on: [push]

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



# Events



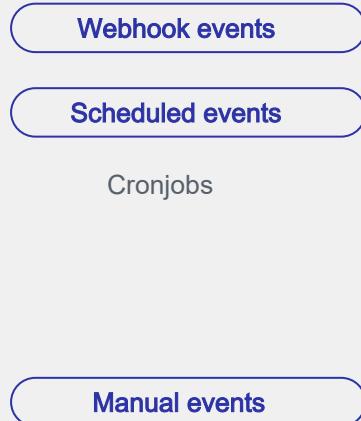
The screenshot shows a GitHub repository interface with the following configuration file content:

```
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!"
```

A red dashed box highlights the `on: push` section of the YAML file. A red arrow points from the **Events** box in the diagram to this highlighted section.



# Events



The screenshot shows the GitHub interface. On the left, there's a sidebar with "Repository" and "Organization" sections. The main area displays a file named "simple-ci.yaml". A red arrow points from the "Scheduled events" category in the diagram to the "on: schedule:" section of the YAML code. The code is as follows:

```
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

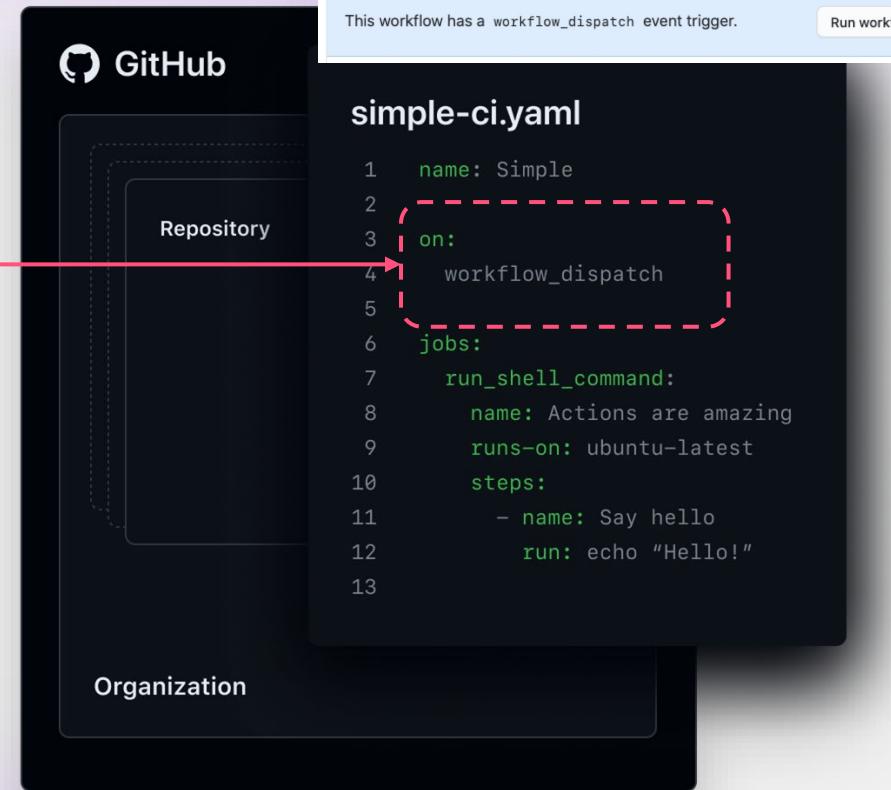
Webhook events

Scheduled events

Manual events

- workflow\_dispatch
- repository\_dispatch

Events





# Runners



GitHub Hosted Runners



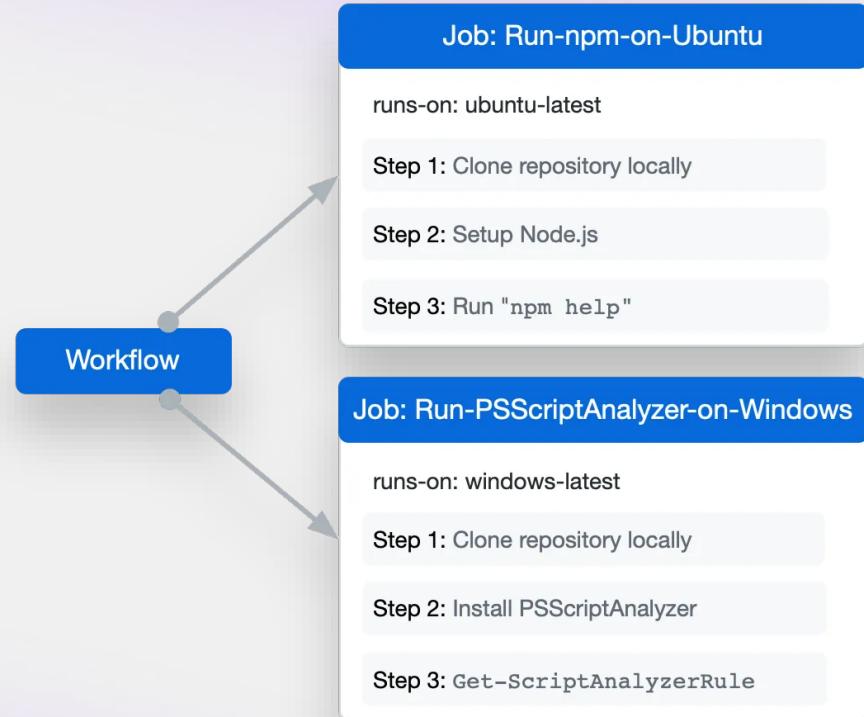
Self-Hosted Runners





# 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	...
<b>Default</b> ⓘ All repositories, excluding public repositories	100	
<b>azure</b> Selected repositories (18), excluding public repositories	260	...
<b>ui-tests</b> Selected repositories (3), excluding public repositories	92	...
<b>mobile</b> Selected repositories (1), excluding public repositories	51	...
<b>aws</b> Selected repositories (10), excluding public repositories	104	...
<b>dotcom</b> Selected repositories (15), excluding public repositories	187	...

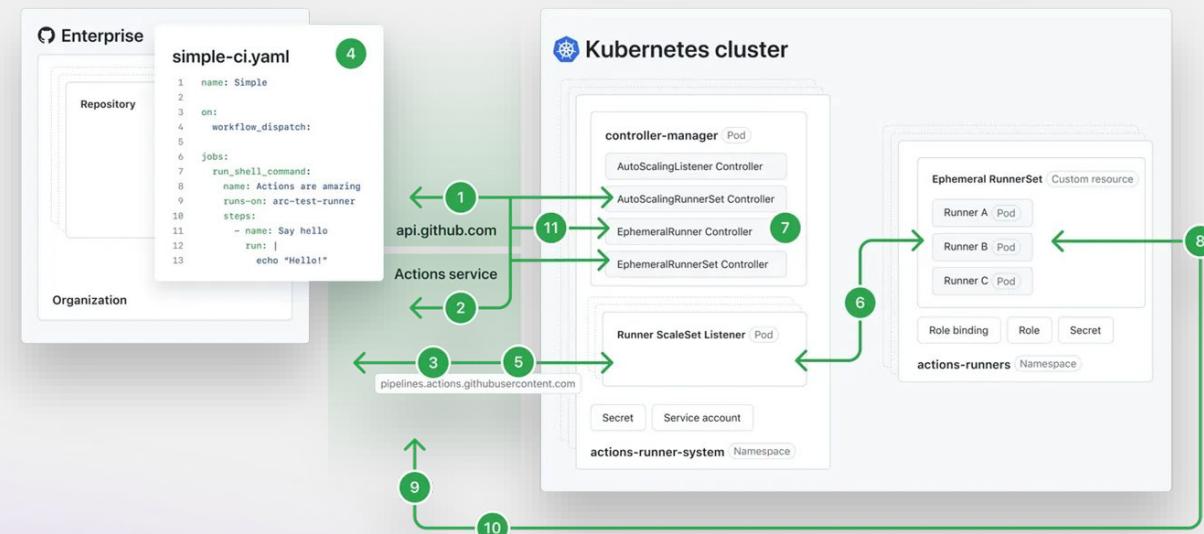
## Shared by the Enterprise

Group	Runners	...
<b>Default</b> All repositories, excluding public repositories	230	
<b>enterprise-hourly</b> All repositories, excluding public repositories	89	
<b>kubernetes</b> All repositories, excluding public repositories	452	



# Self-Hosted Runners - ARC

- ✓ 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 GitHub Enterprise architecture. It shows three nested layers: 'Organization' at the outermost level, 'Repository' in the middle, and a dashed-line box representing the scope of a workflow file ('simple-ci.yaml'). To the right of the diagram is the content of the 'simple-ci.yaml' file.

```
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 the GitHub Marketplace interface. At the top, there's a search bar with the query "sort:popularity-desc". Below it, a sidebar lists categories: Types (Actions selected), Apps, and Stacks. The main area is titled "Actions" and contains a sub-header: "An entirely new way to automate your development workflow." It shows 15831 results for "sort:popularity-desc" filtered by Actions. A blue callout box at the bottom right of the page says "20k+ Actions available in the GitHub Marketplace".

Category	Action	Description	Stars
API management	TruffleHog OSS	By trufflesecurity Scan Github Actions with TruffleHog	9.7k
Chat	Super-Linter	By github It is a simple combination of various linters, written in bash, to help validate your source code	8.3k
Code quality			
Code review			
Continuous integration			
Dependency management			
Deployment			
IDEs			
Learning			
Localization			
Mobile			



# 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 a web browser window with the GitHub URL in the address bar. The main content is titled "Choose a workflow" with the subtitle "Build, test, and deploy your code. Make code reviews, branch management, and issue triaging work the way you want. Select a workflow below." Below this is a link "Skip this and set up a workflow yourself →".  
  
On the left, there's a sidebar with "Categories" (Automation, Continuous integration, Deployment, Security) and a search bar "Search workflows".  
  
The main area displays "Found 37 workflows" in a grid format. Each card includes the workflow name, provider, a brief description, and two buttons: "Configure" and "Code scanning".

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



# 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

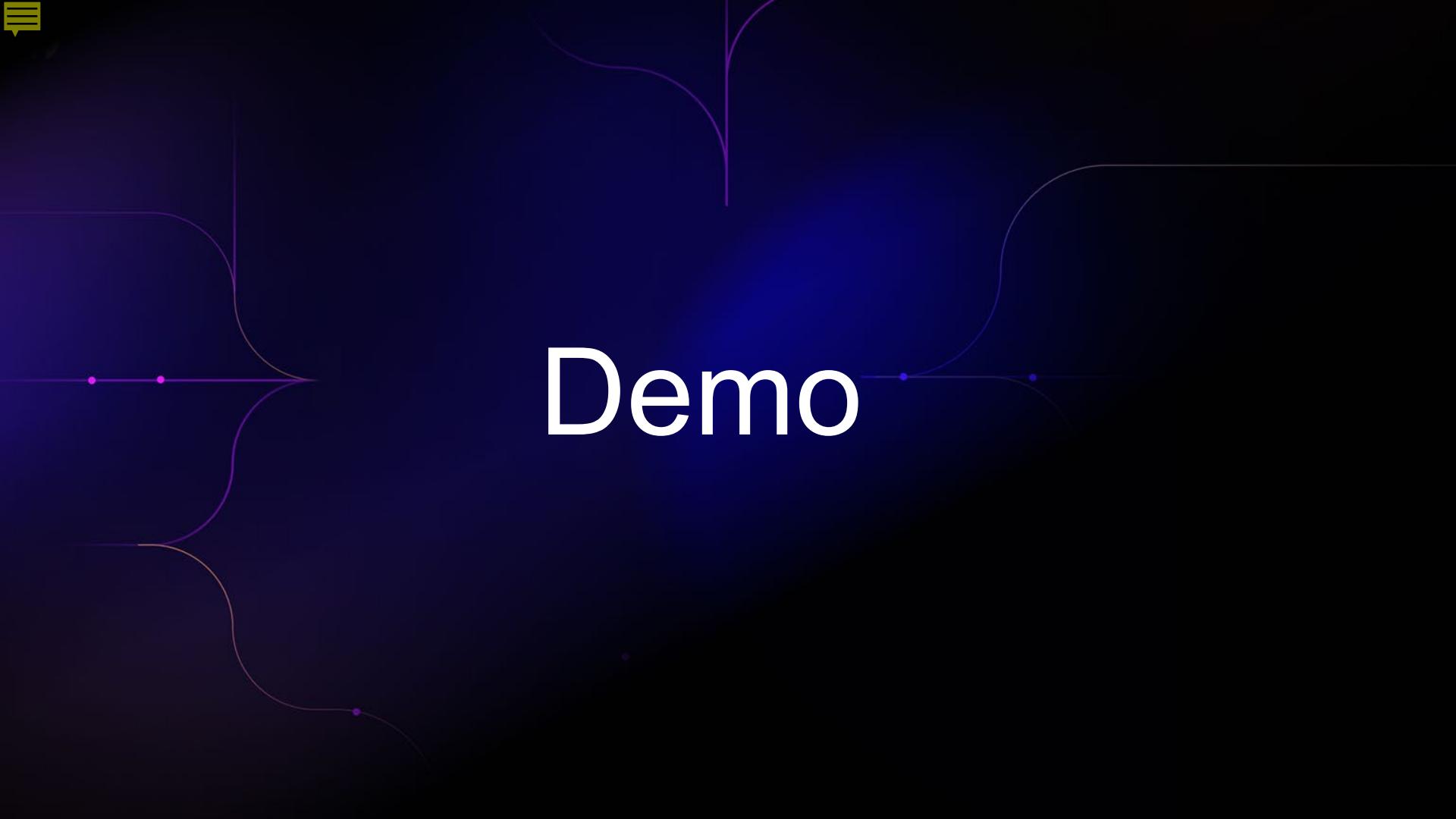
## toJSON/fromJSON

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>	<b>GHEC 180</b>  32 core: <b>1,500</b> concurrent jobs  64 core: <b>1,900</b> concurrent jobs	50 maximum concurrent macOS jobs
Job Execution	<b>6</b> hours	On error, jobs terminated and seen as failed. This limit <b>doesn't</b> apply to self-hosted runners
Workflow Run	<b>72</b> hours	On error, workflow cancelled when reached. This limit also applies to self-hosted runners
Job Queue	<b>24</b> hours	Jobs terminated when the limit is reached. This limit <b>only</b> applies to self-hosted runners
API Requests	<b>1000</b> per hour per repo	On error, API calls fail. Can cause jobs to fail. This limit also applies to self-hosted runners
Job Matrix	<b>256</b> per run	This limit also applies to self-hosted runners



Demo





# 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

The screenshot shows the GitHub Environments configuration interface for the 'Production' environment. At the top, there's a navigation bar with a back arrow, forward arrow, and a lock icon for 'github.com'. Below it, the title 'Environments / Configure Production' is displayed. A section titled 'Deployment protection rules' is shown with the sub-section 'Required reviewers' checked. It includes a note about specifying approvers and a search bar for people or teams. Other options like 'Prevent self-review' and 'Wait timer' are also listed. A beta feature 'Enable custom rules with GitHub Apps' is mentioned with links to existing apps and creating your own rules. Finally, a checkbox for 'Allow administrators to bypass configured protection rules' is checked, and a 'Save protection rules' button is present.

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

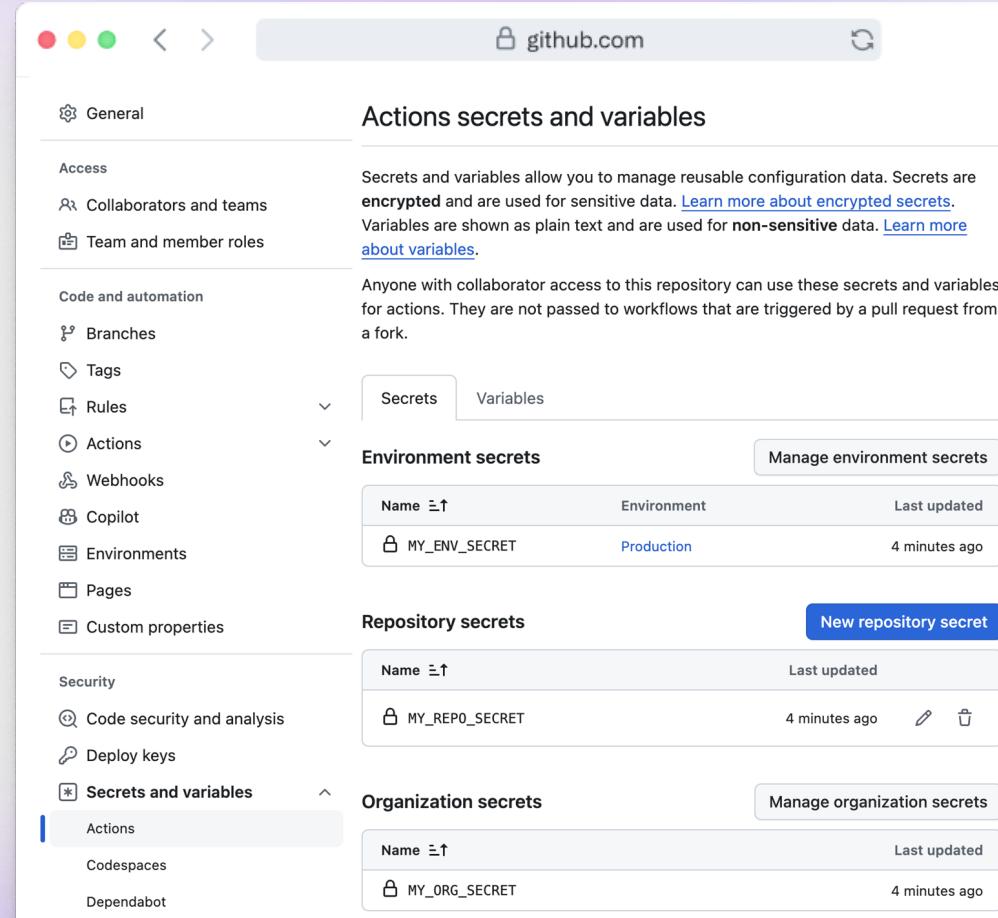
## Integration & Management

- 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 Actions secrets and variables settings page. The left sidebar lists various settings categories: 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. The right side has three main sections: Environment secrets, Repository secrets, and Organization secrets. Each section lists secrets with columns for Name, Environment, Last updated, and manage buttons.

Name	Environment	Last updated
MY_ENV_SECRET	Production	4 minutes ago

Name	Last updated
MY_REPO_SECRET	4 minutes ago

Name	Last updated
MY_ORG_SECRET	4 minutes ago



# Environment secrets

- ✓ Built-in secret store

- ✓ Encrypted (Lib Sodium 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' page with a sidebar and three main sections: Environment secrets, Repository secrets, and Organization secrets.

**Environment secrets:** This section is highlighted with a red box. It contains a table with one row:

Name	Environment	Last updated
MY_ENV_SECRET	Production	4 minutes ago

**Repository secrets:** This section is highlighted with a red box. It contains a table with one row:

Name	Last updated	Actions
MY_REPO_SECRET	4 minutes ago	

**Organization secrets:** This section is highlighted with a red box. It contains a table with one row:

Name	Last updated	Actions
MY_ORG_SECRET	4 minutes ago	

**Sidebar:** The sidebar on the left lists various repository settings: 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 expanded), Actions, Codespaces, and Dependabot.

# Using secrets in workflows



All secrets can be accessed using the same syntax;

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

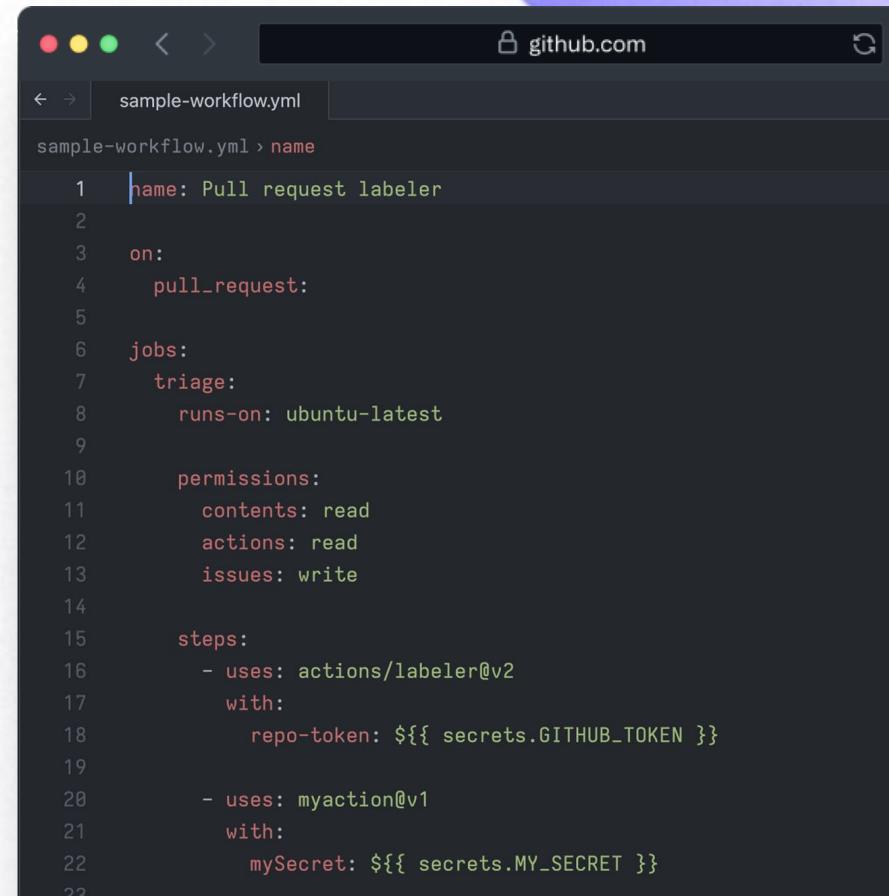


Every workflow run provisions a `GITHUB_TOKEN` secret by default

- 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
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   contents: read
12   actions: read
13   issues: write
14
15 steps:
16   - uses: actions/labeled@v2
17     with:
18       repo-token: ${{ secrets.GITHUB_TOKEN }}
19
20   - uses: myaction@v1
21     with:
22       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](https://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  Allow GitHub Actions to create and approve pull requests
36
37 Save
```



# Demo



# 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 settings. 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 "All repositories" dropdown is set to "Allow all actions and reusable workflows".

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

**Artifact and log retention:** Choose the default repository settings for artifacts and logs. The "90 days" button is selected.

**Fork pull request workflows from outside collaborators:** Choose which subset of outside collaborators will require approval to run workflows. The "Require approval for first-time contributors who are new to GitHub" radio button is selected.



# 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



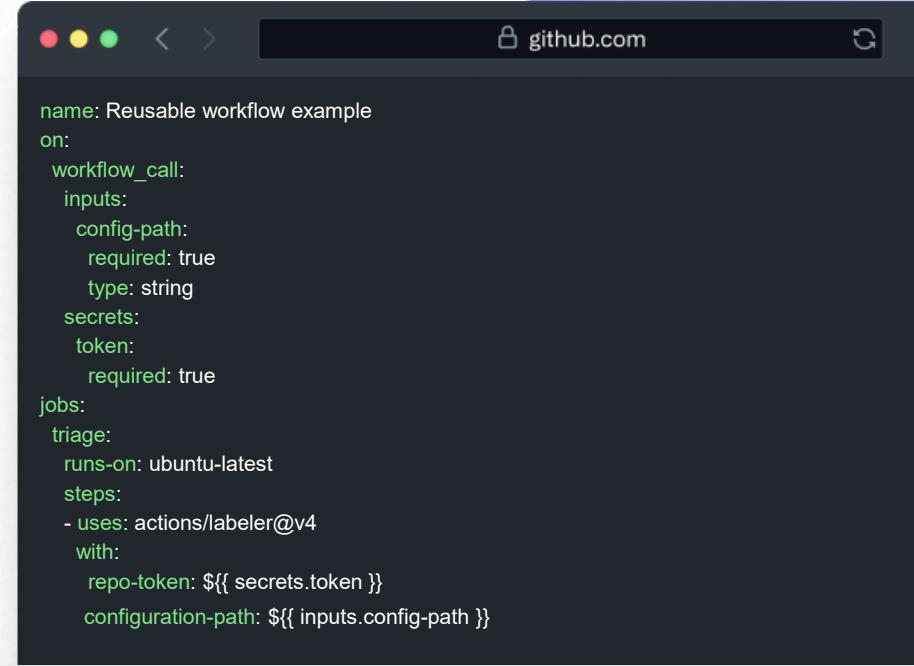
## Centralized Management

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

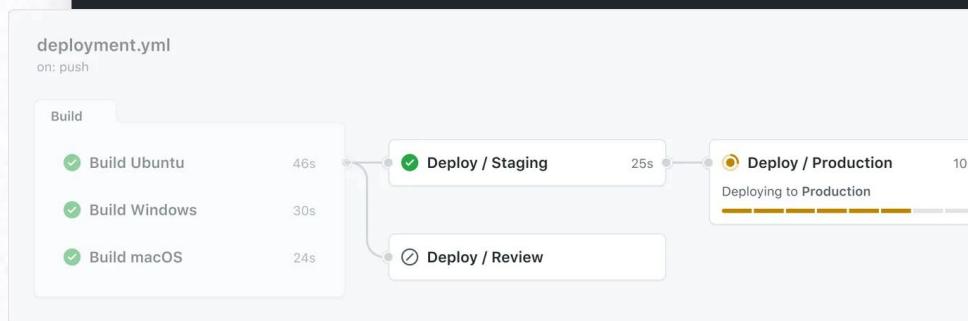


# Reusing workflows

- ✓ 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/llabeler@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



# Demo

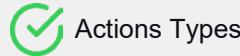


# Building Actions

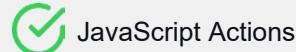
## Build Actions & Workflows



# Write Custom Action



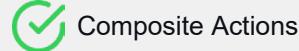
Actions Types



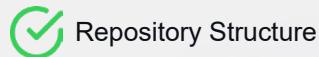
JavaScript Actions



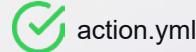
Docker Actions



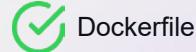
Composite Actions



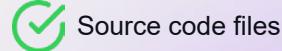
Repository Structure



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"
```



# 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](https://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"]
```



# Composite Action

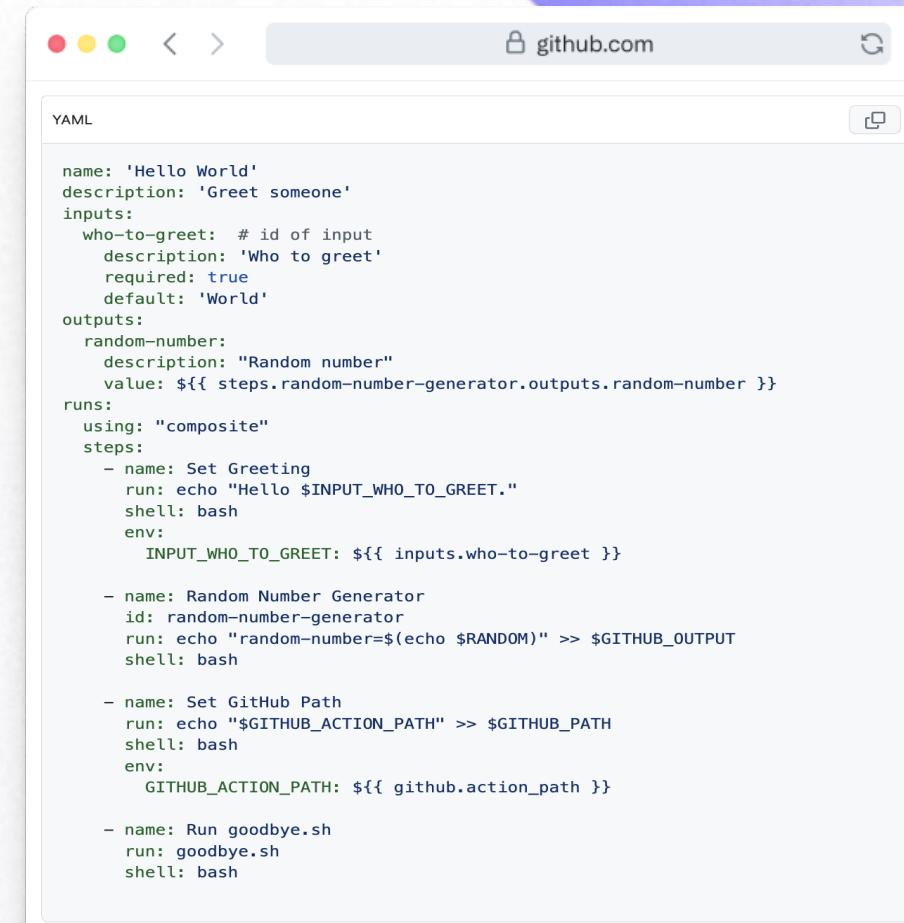
A Composite Action in GitHub is an action that combines multiple individual steps into one reusable component. Instead, they use YAML syntax to define a sequence of steps that are executed together as a single unit. This allows you to encapsulate common tasks and reuse them across different workflows, 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"), and runs multiple steps. The first step sets a greeting message, the second generates a random number, the third sets the GitHub path, and the fourth runs 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
```



# 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

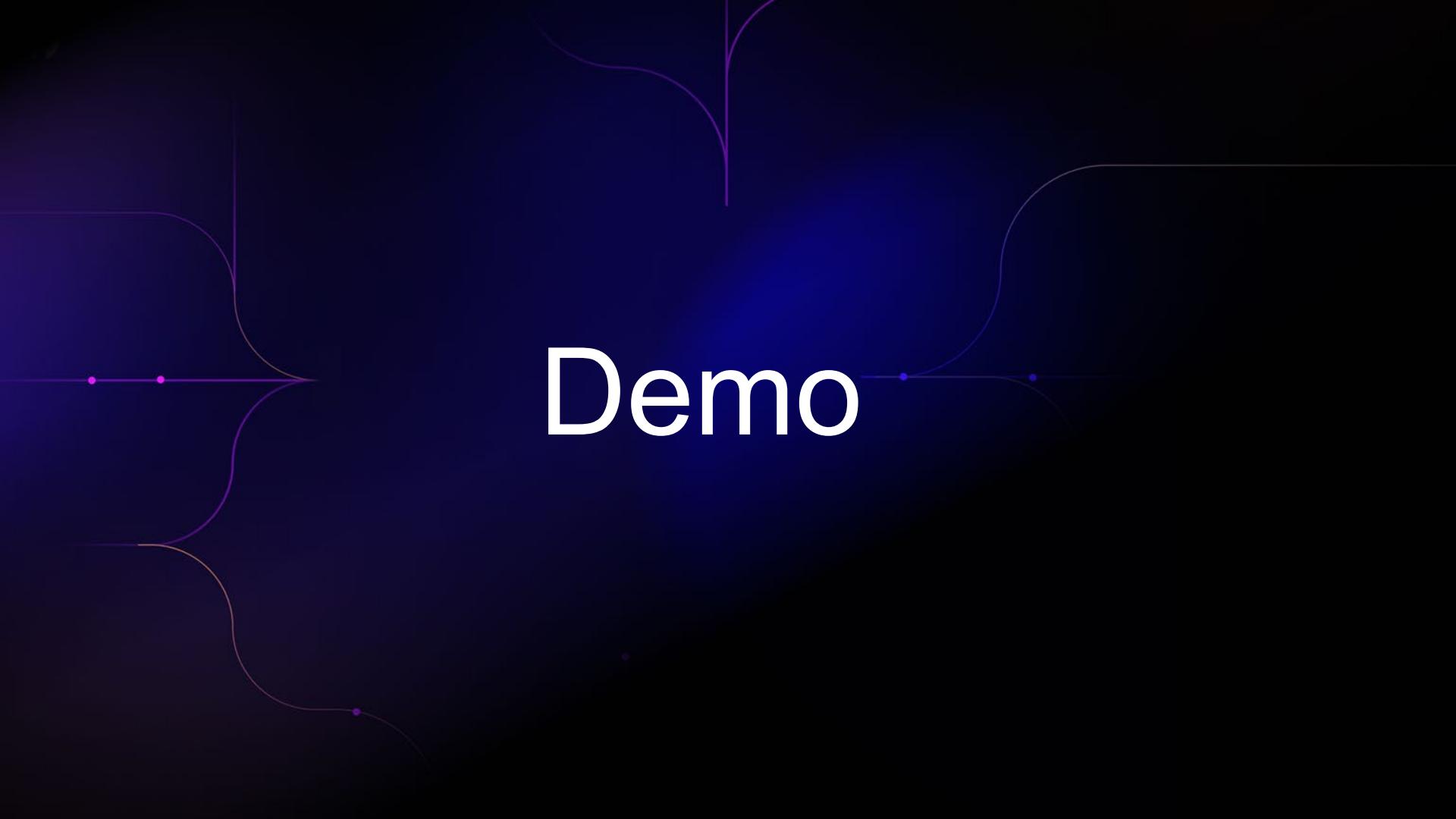
- Local Testing
- Validation

### Reusability

- Reusable Components
- Composite Actions
- Use GitHub Actions Toolkit

### Optimization

- Efficient Code
- Caching



# Demo



# 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 titled "Bash" with the URL "github.com" in the address bar. The window contains two numbered steps:

- 1 Install the GitHub Actions Importer CLI extension:  
A command is shown in the terminal: `gh extension install github/gh-actions-importer`.
- 2 Verify that the extension is installed:  
The terminal displays the help output for the `actions-importer` command, listing various subcommands and their descriptions.

```
$ 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

Travis CI

CircleCI

GitLab CI

Azure

Bitbucket

Bamboo

TeamCity

A screenshot of a web browser window displaying the GitHub Actions Importer documentation. The URL in the address bar is [github.com](https://github.com). The page content shows the usage of the command `$ gh actions-importer migrate -h`. It includes a detailed description of the command: "Convert a pipeline to a GitHub Actions workflow and open a pull request with the changes." Below the description is a section titled "Commands:" which lists several sub-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), and `travis-ci` (Convert a Travis CI pipeline to a GitHub Actions workflow and open a pull request with the changes). Ellipses [...] are shown after the first command in the list.

```
$ 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 Self-Hosted Runners interface. At the top, there's a header with the GitHub logo, the page title 'Runners / Self-Hosted Runners', and a search bar. Below the header, the main content area has several sections:

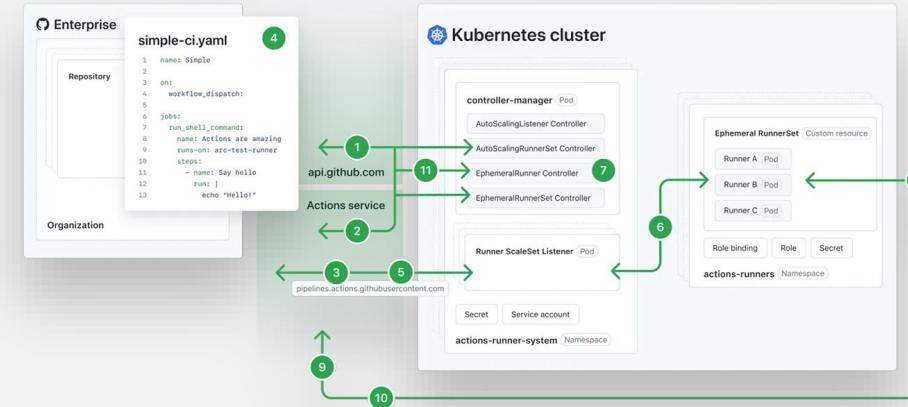
- Add new self-hosted runner**: A section with instructions and links to GitHub Terms of Service or GitHub Corporate Terms of Service. It includes dropdowns for 'Runner image' (selected 'macOS'), 'Architecture' (selected 'x64'), and a 'Download' section containing shell commands to download and install the runner.
- Configure**: A section with shell commands to create a runner and start the configuration experience.
- Using your self-hosted runner**: A section with a YAML example for a workflow file.
- Policies**, **Runners** (selected), **Runner groups**: A navigation bar.
- Includes all runners across self-hosted and GitHub-hosted runners.**: A note.
- Search runners**: A search bar.
- Runners**: A list of runners:
  - Standard GitHub-hosted runners**: Ready-to-use runners managed by GitHub. [Learn more.](#)
  - ubuntu-latest-m**: ubuntu-latest-m. Runner group: Default Larger Runners. Public IP: **Disabled**.
- New GitHub-hosted runner**: Pay-as-you-go, customizable, secure, scaled & managed by GitHub.
- New self-hosted runner**: Bring your own infrastructure.



# 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 browser window with the URL [github.com](https://github.com). The page displays a GitHub Action workflow file named `main.yml`. The code defines a deployment step that depends on a build step, runs on an Ubuntu latest runner, and includes several steps to log in to Azure, set up kubectl, and get AKS credentials before finally deploying the application using kubectl apply commands.

```
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
```



# What's Next

- ✓ OIDC with GitHub Actions
- ✓ Self-hosted runners with private vNet
- ✓ Deployment Environment
- ✓ Deployment Environment RuleSets



Thanks