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
Tirage Issues and Project Boards using Actions

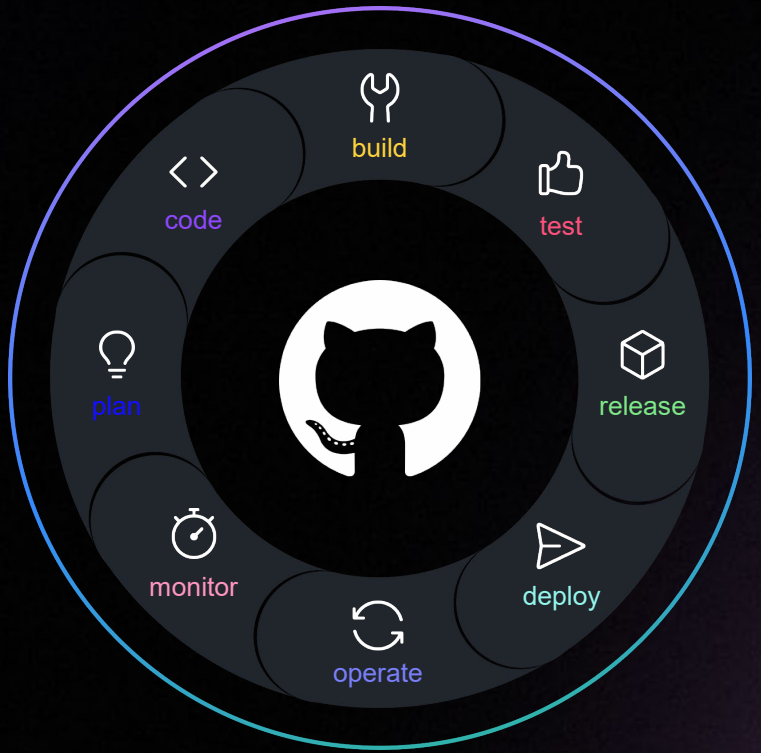
Code
Automates code builds, linting upon pull request creation

Build
Compile and build, Store artifacts

Test
Automotion of the unit tests, integration tests

Release
Publish release, Release notes, Versioning, and Upload artifacts

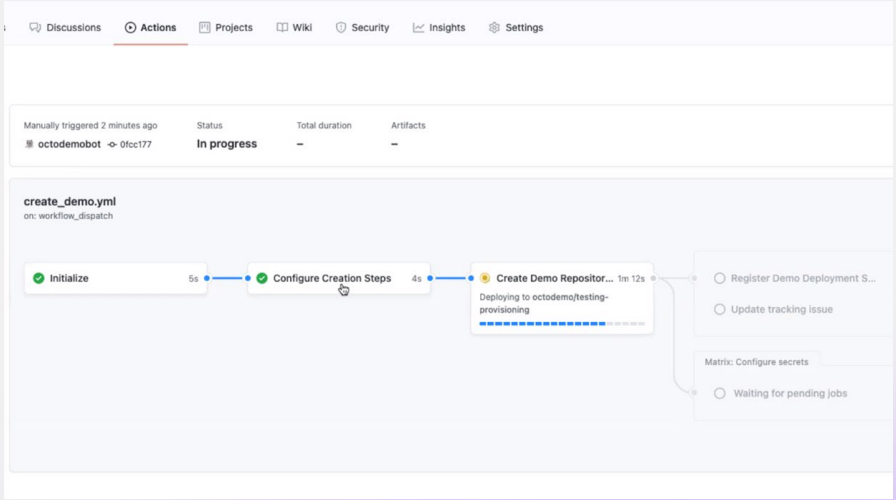
Deploy
Deploy applications to your chosen environment



Automation accessible to every developer

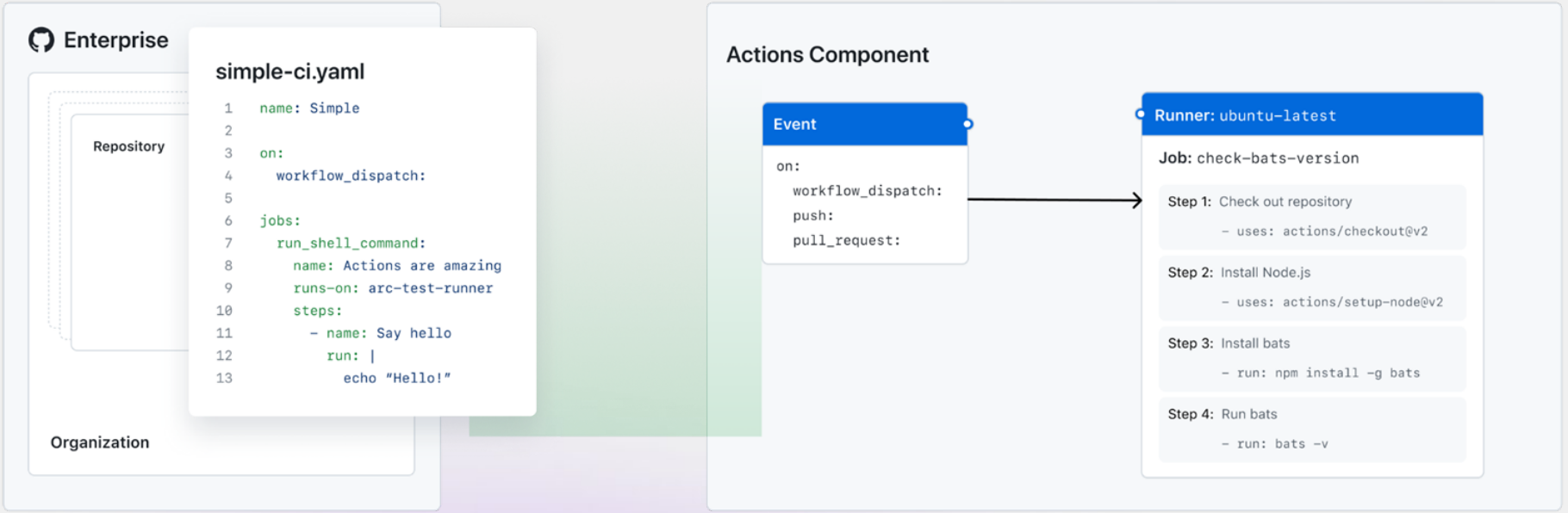
Actions

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



- 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



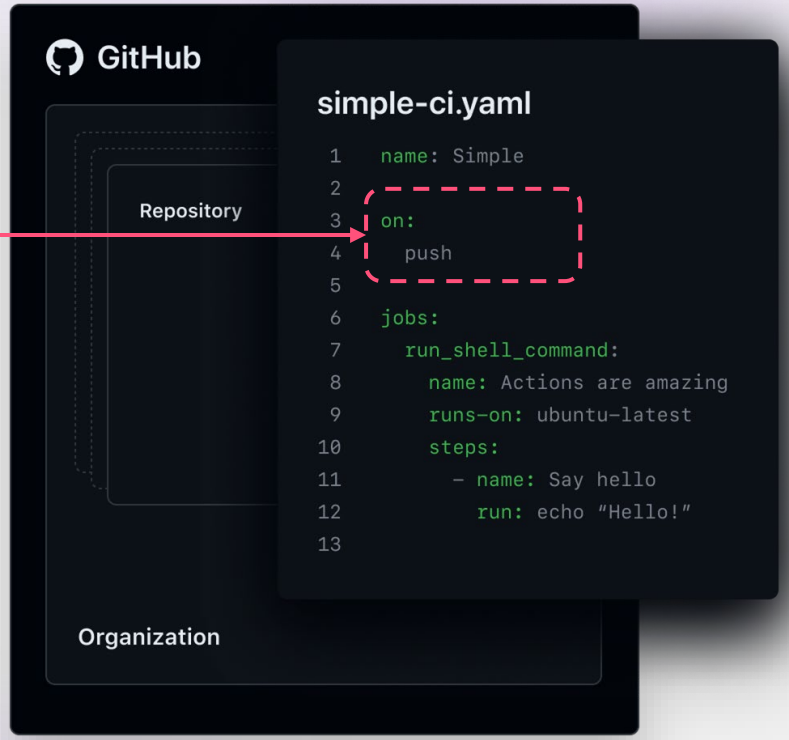
Actions Workflow Syntax

Understanding the workflow file

The **name of the workflow** as it will appear in the "Actions" tab of the GitHub repository	`name: learn-github-actions`
The **name for workflow runs** generated from the workflow, which will appear in the list of workflow runs on your repository's "Actions" tab	`run-name: ${{ github.actor }} is learning GitHub Actions`
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	`on: [push]`
Groups together all the **jobs** that run in the learn-github-actions workflow.	`jobs:`
Defines a job named **check-bats-version**. The child keys will define properties of the job.	`  check-bats-version:`
Configures the **job runner** on the latest version of an Ubuntu Linux runner.	`    runs-on: ubuntu-latest`
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	`    steps:` `      - uses: actions/checkout@v4`

Events

- **Webhook events**
  - Pull Request
  - Push to repo
  - Issues
  - Releases
  - ...
- **Scheduled events**
- **Manual events**



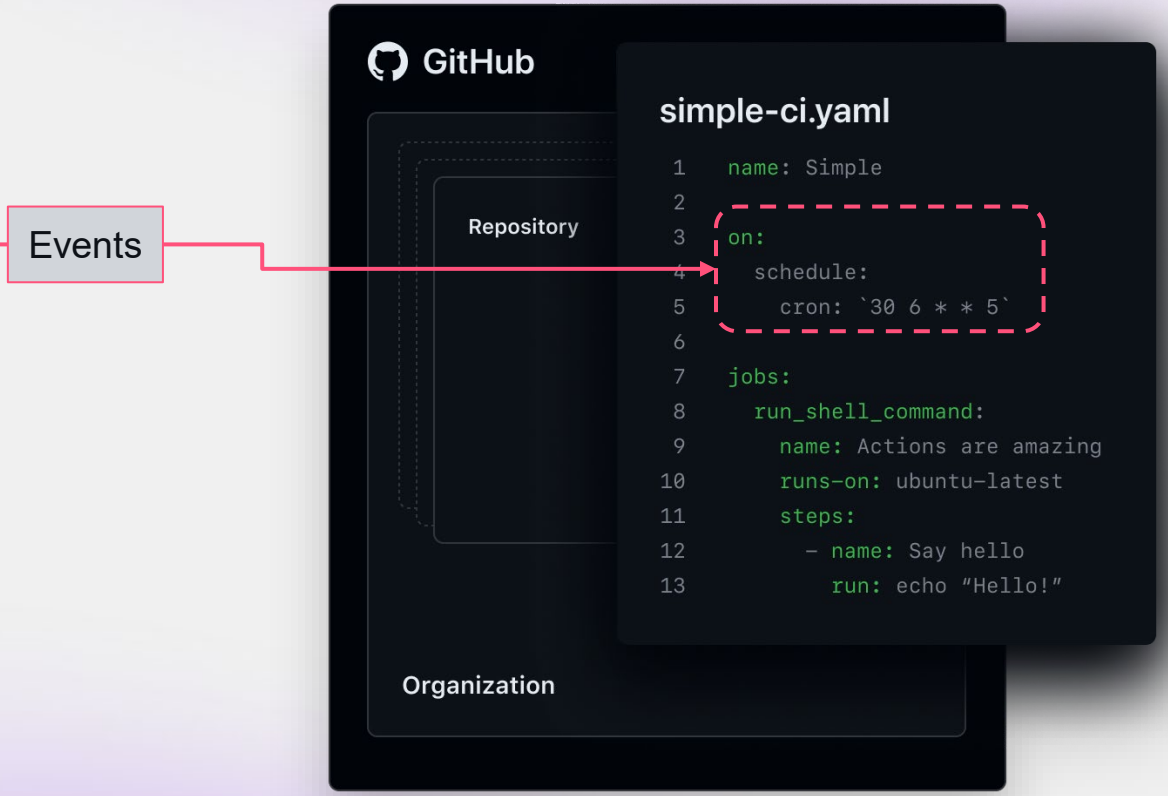
Events

Webhook events

Scheduled events

Cronjobs

Manual events

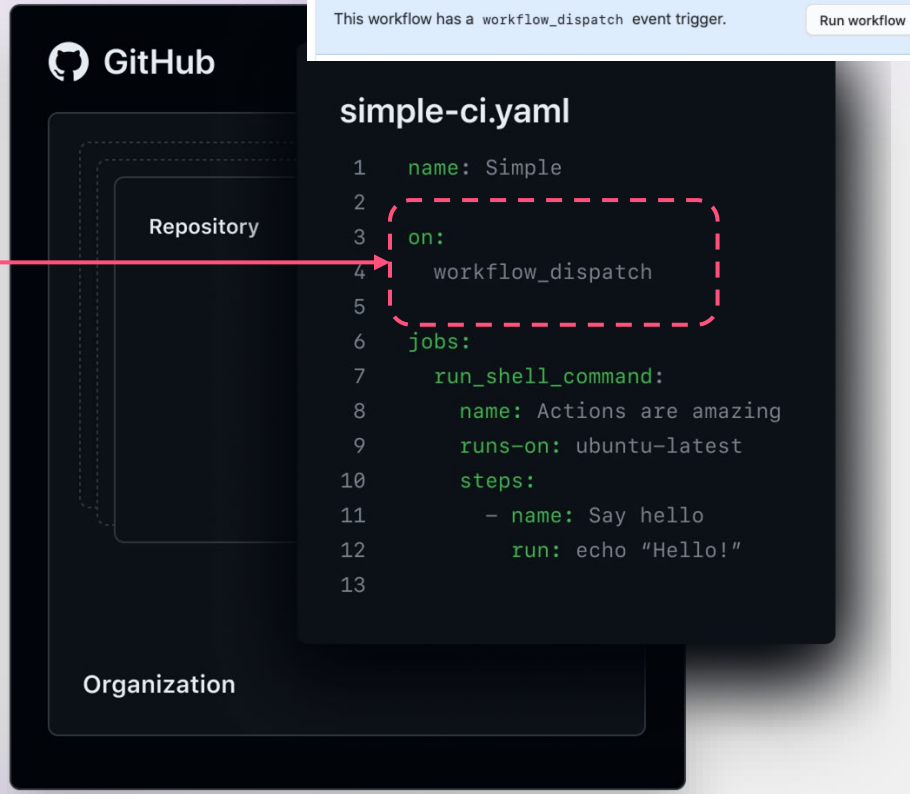


Events

- Webhook events
- Scheduled events
- Manual events

Events

- workflow_dispatch
- repository_dispatch



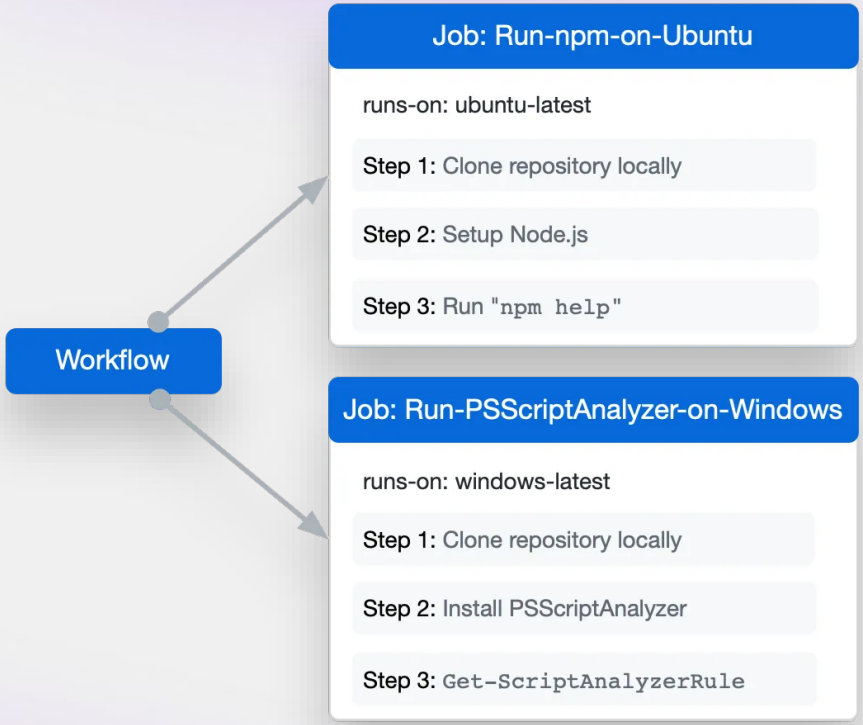
Runners

GitHub Hosted Runners

Self-Hosted Runners

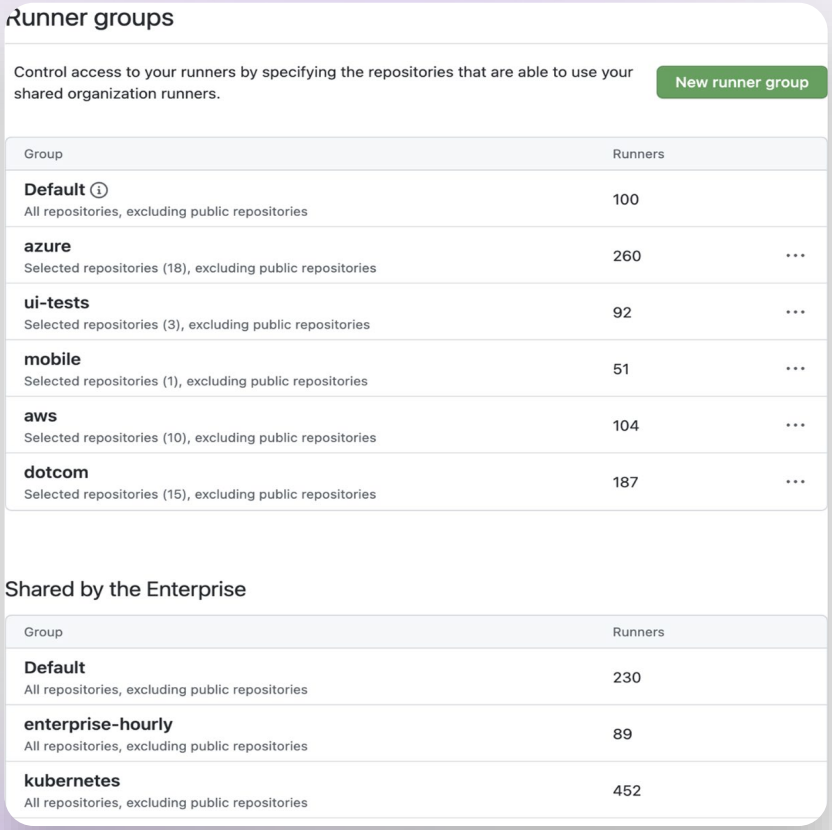
GitHub Hosted Runners

- 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



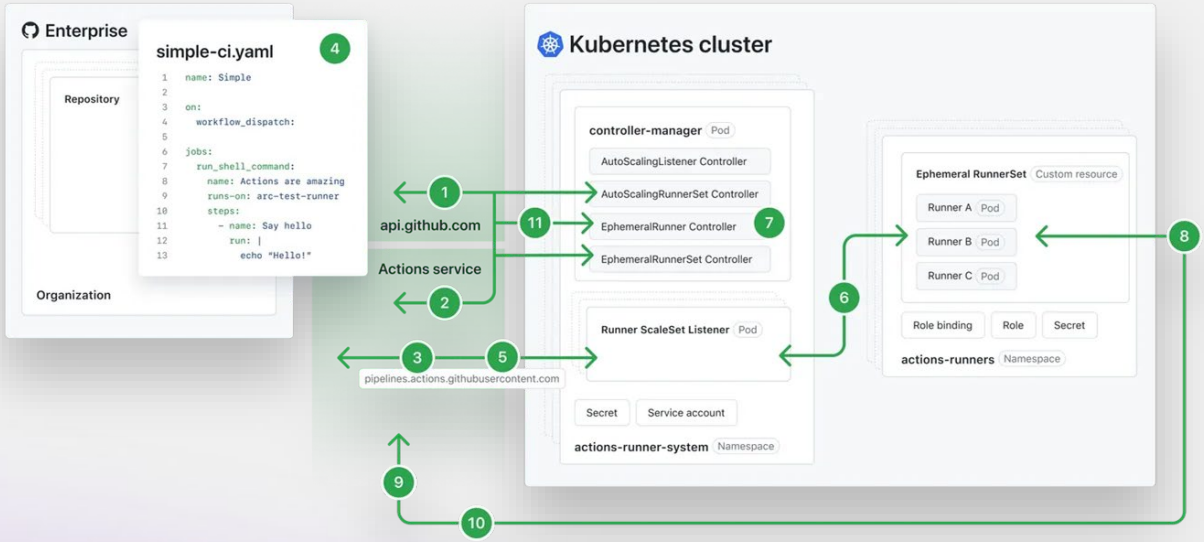
Self-Hosted Runners

- 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



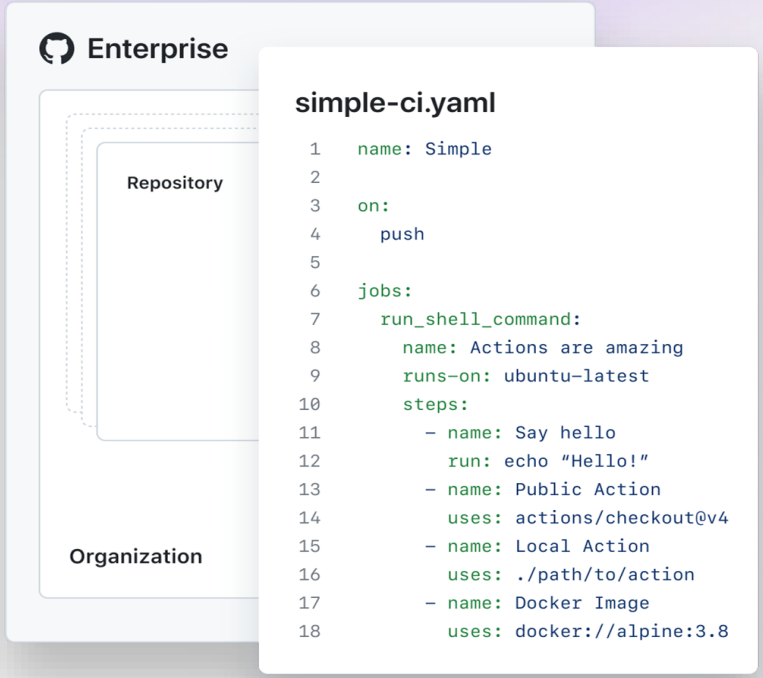
Self-Hosted Runners - ARC

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



Actions

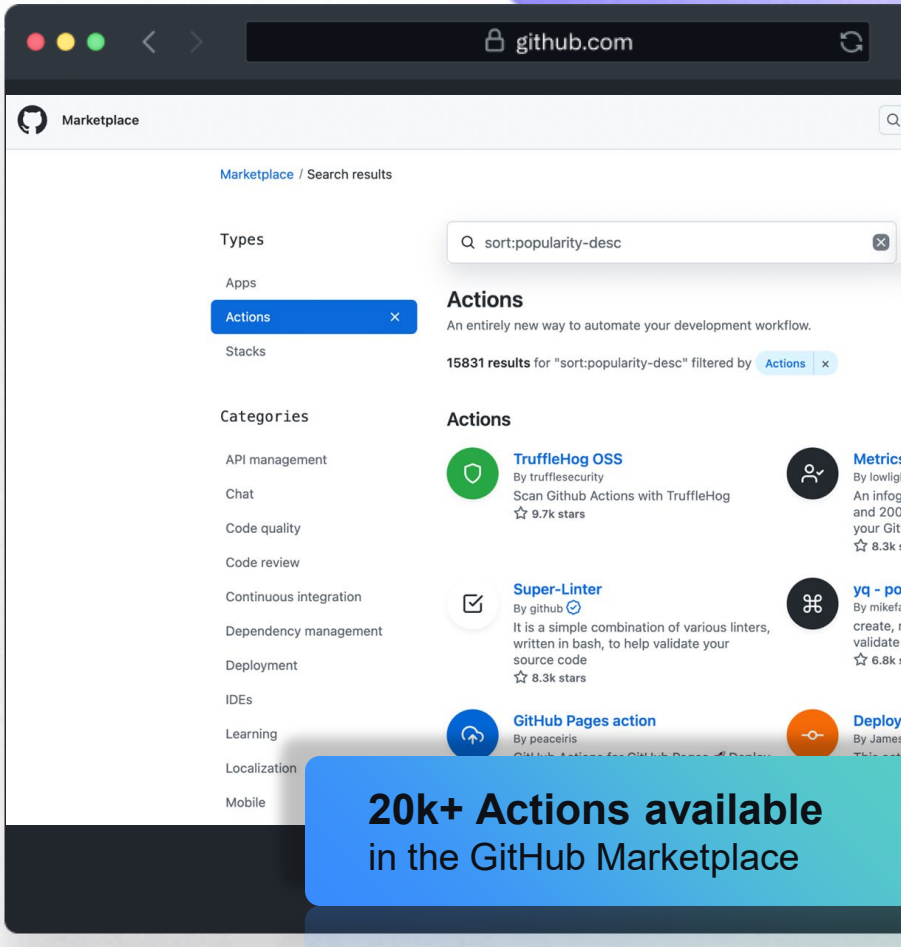
- Reusable units of code that can be referenced in a workflow
- GitHub runs them in Node.js runtime, or in containers
- Reference an Action, or run scripts directly



Script | Public Action | Local Action | Docker Image

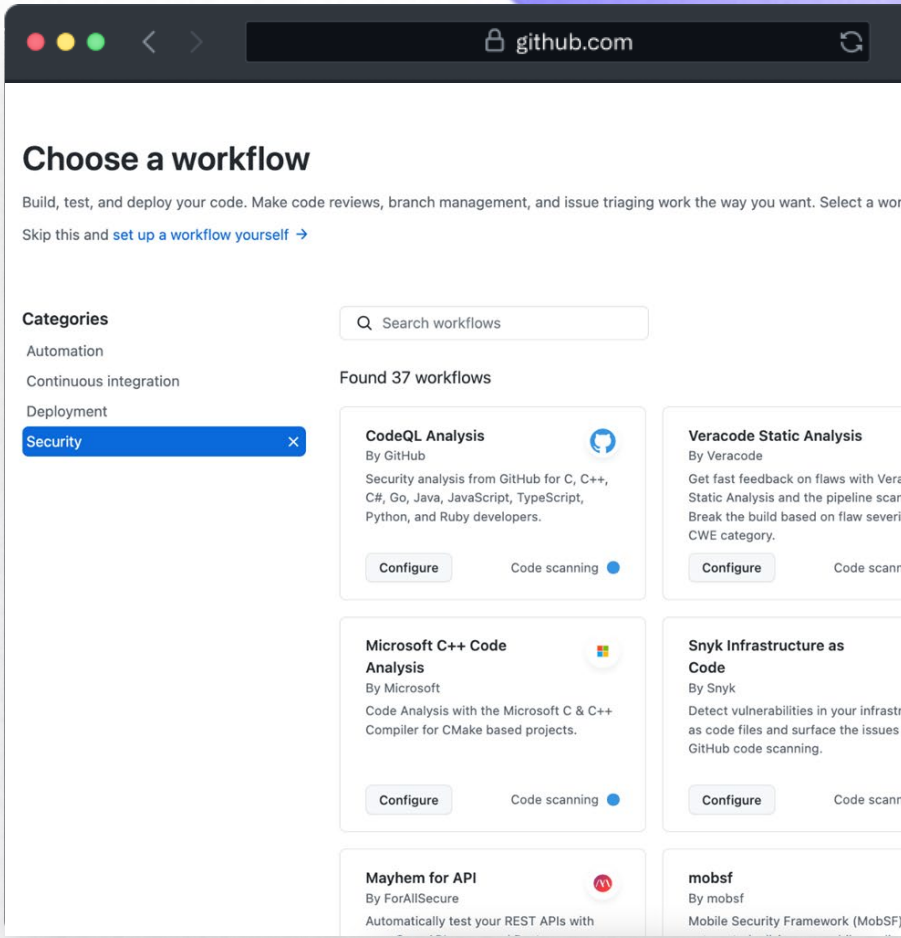
Largest Connected Developer Community

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

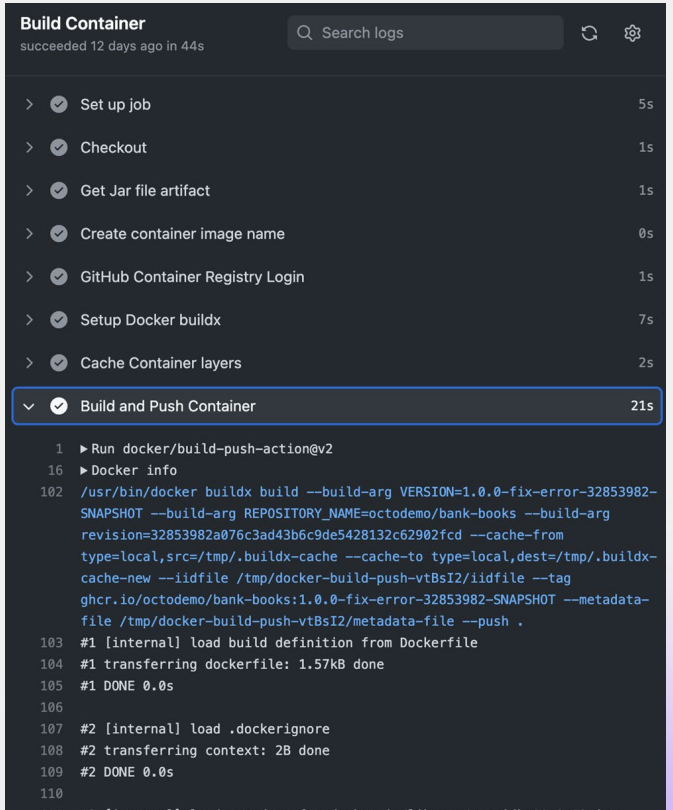


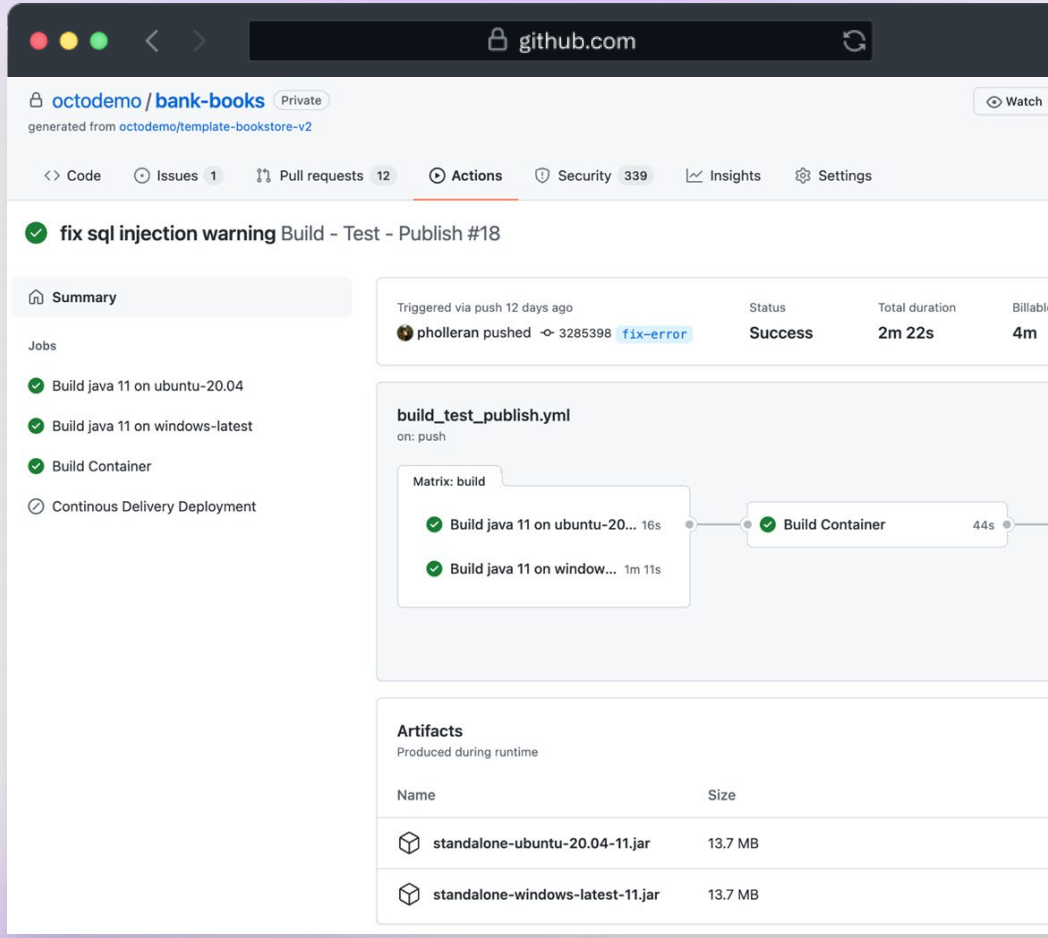
Starter workflows

- 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.



Workflow Logs





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 (Based on Enterprise Plan) This is for the entire enterprise	GHEC 180 32 core: 1,500 concurrent jobs 64 core: 1,900 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	72 hours	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	1000 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

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

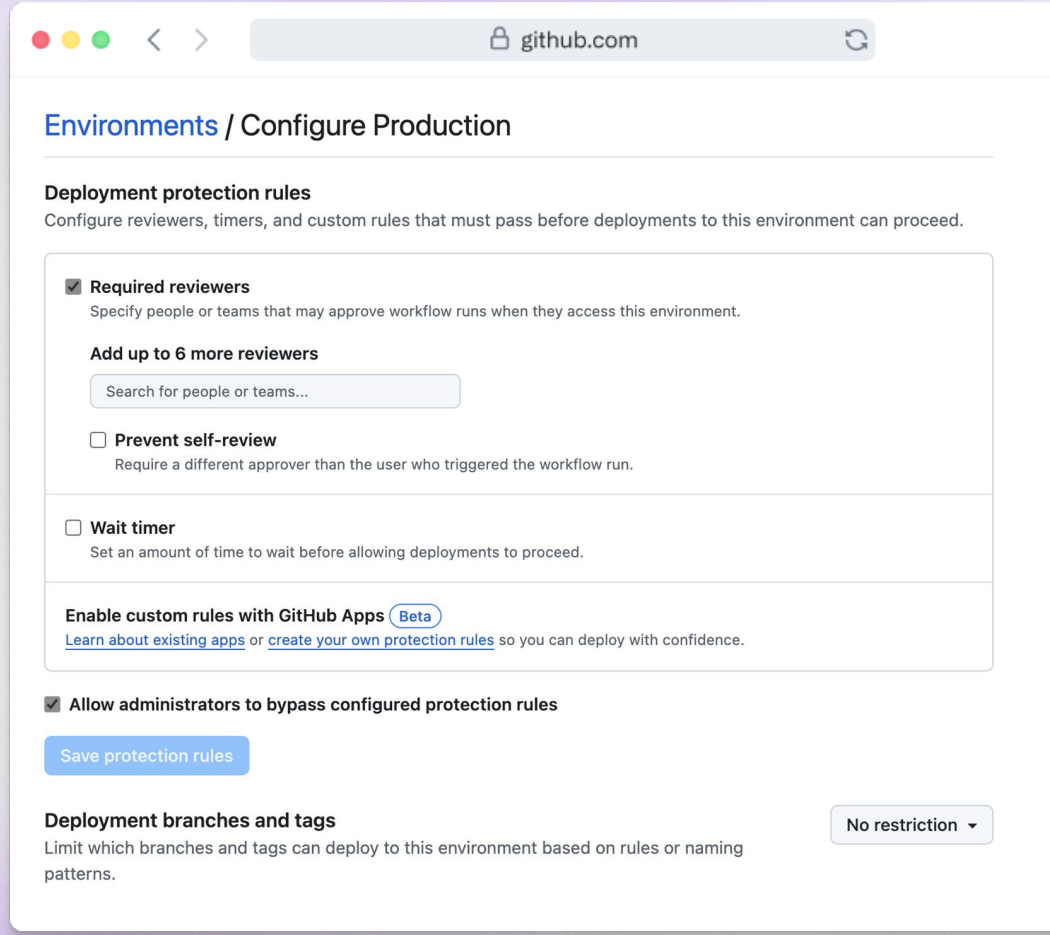
- Support a review and approval process
- Allows for designated reviewers to approve or reject deployments

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



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

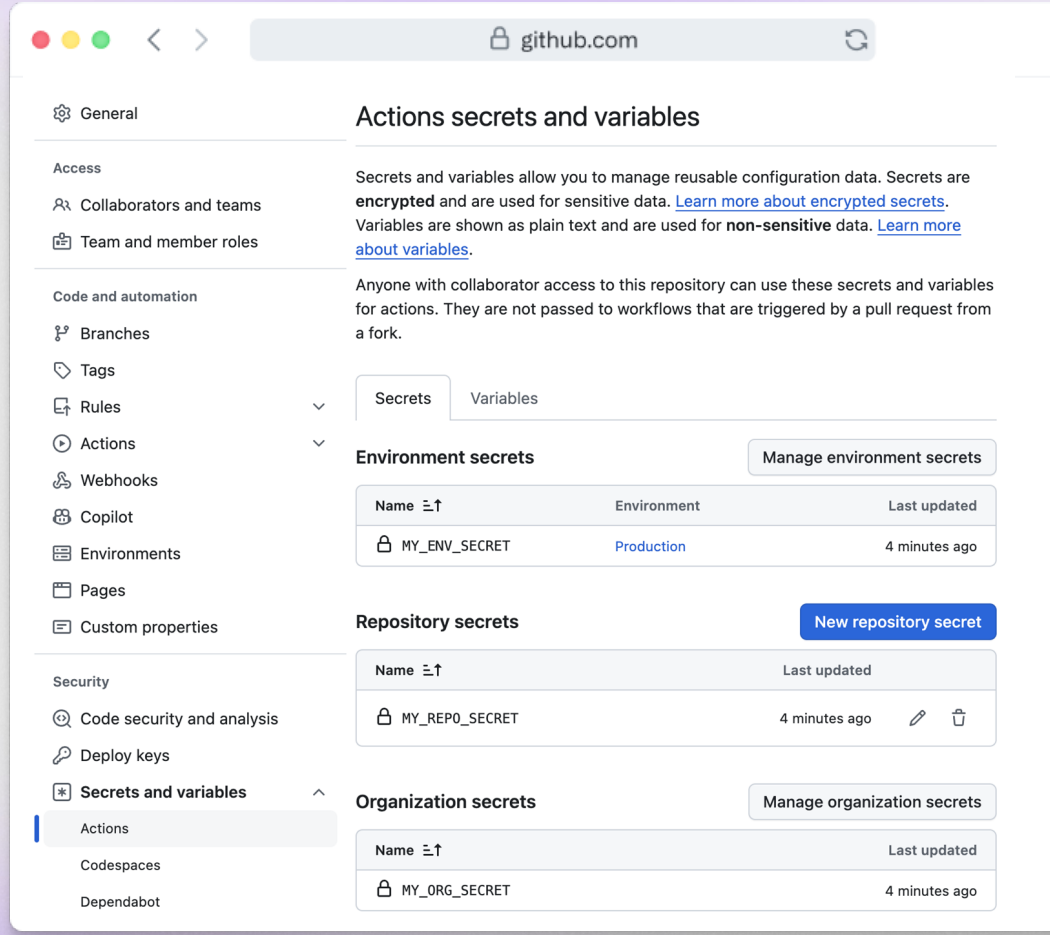
- Secrets are encrypted before storage & during transmission and storage
- Redacts secrets from logs and allows to use short-lived tokens

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



Environment secrets

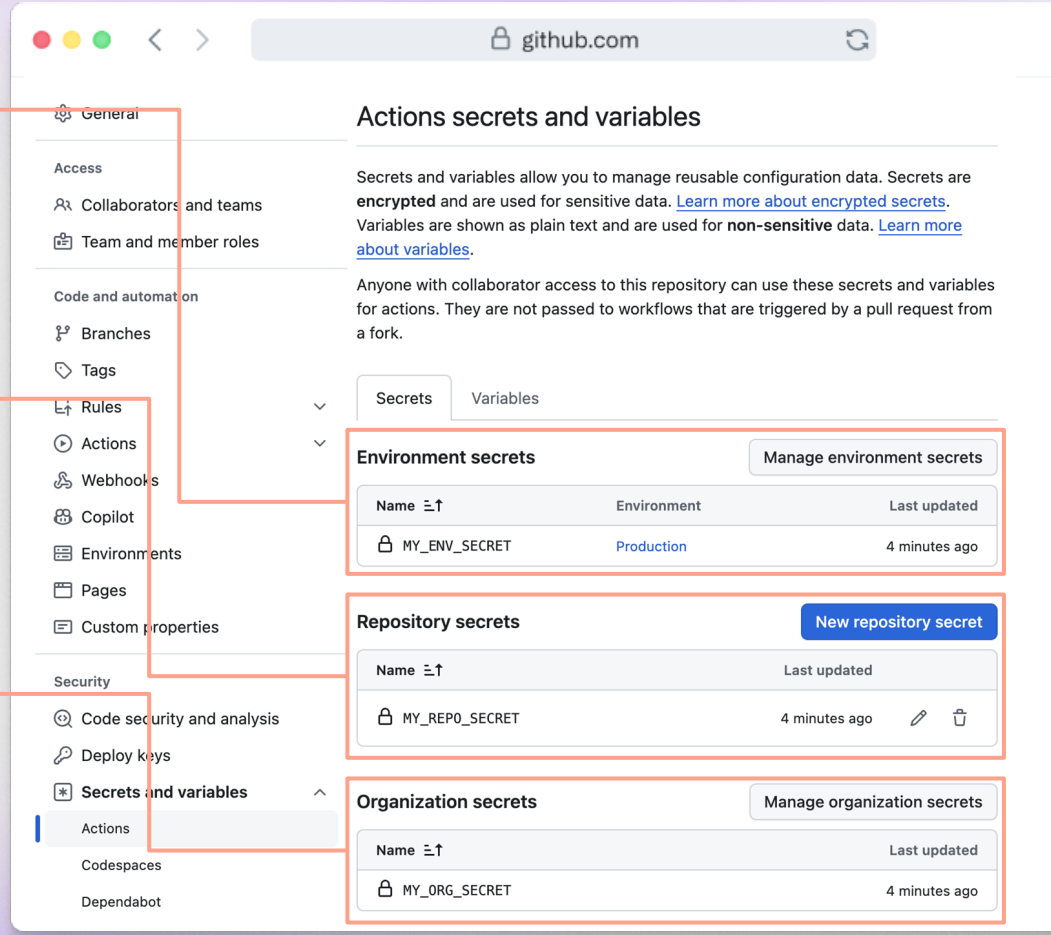
- Built-in secret store
- Encrypted (LibSodium sealed box)

Repository secrets

- Use directly from your workflow
- Redacted in workflow logs

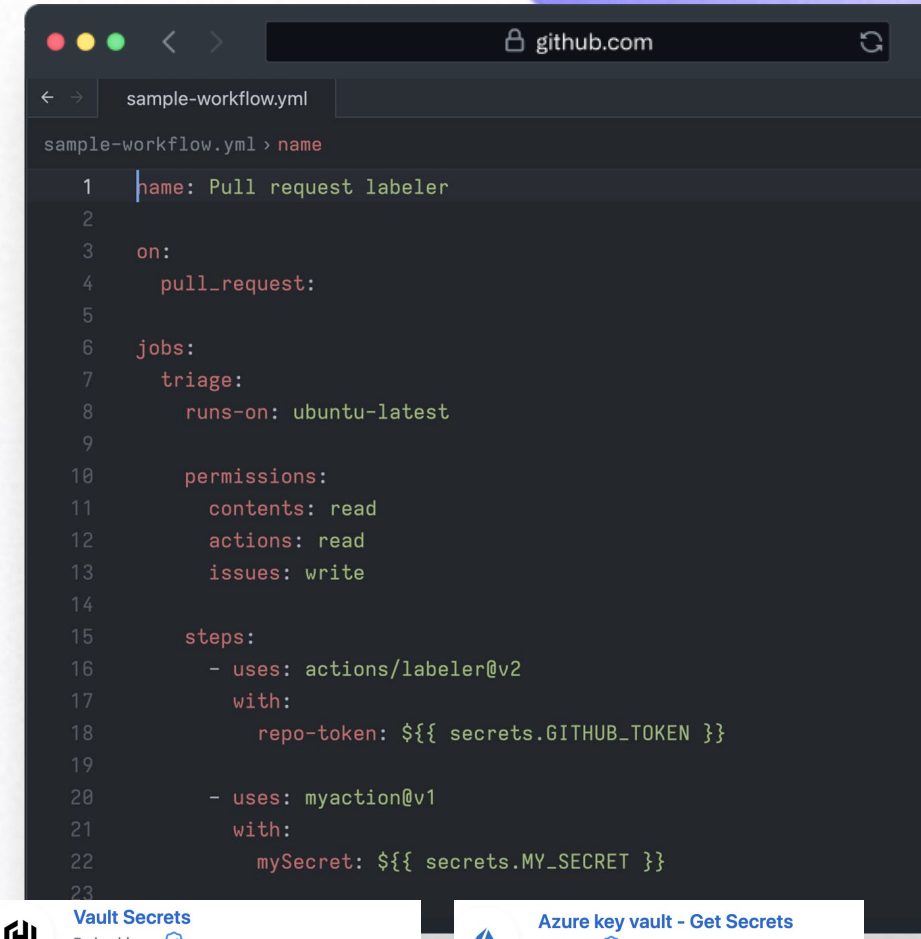
Organization secrets

- API & CLI support
- Organization / repository / environment secrets



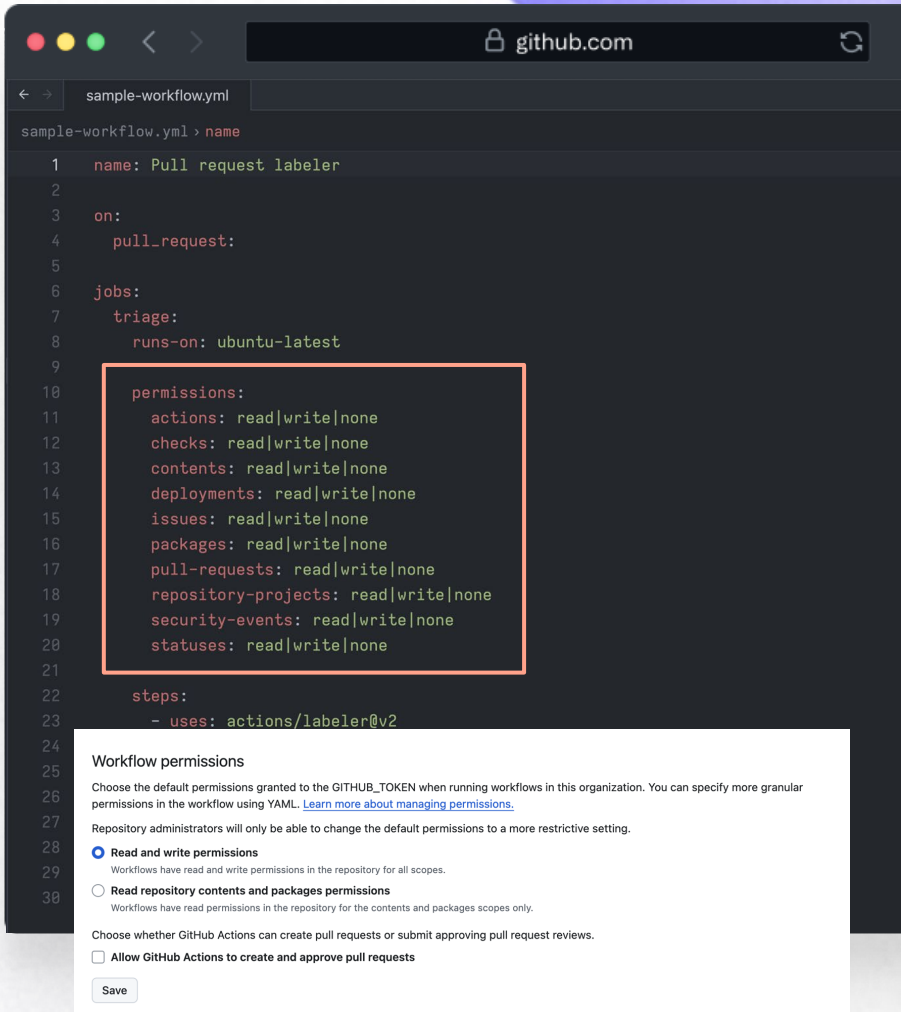
Using secrets in workflows

- 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



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



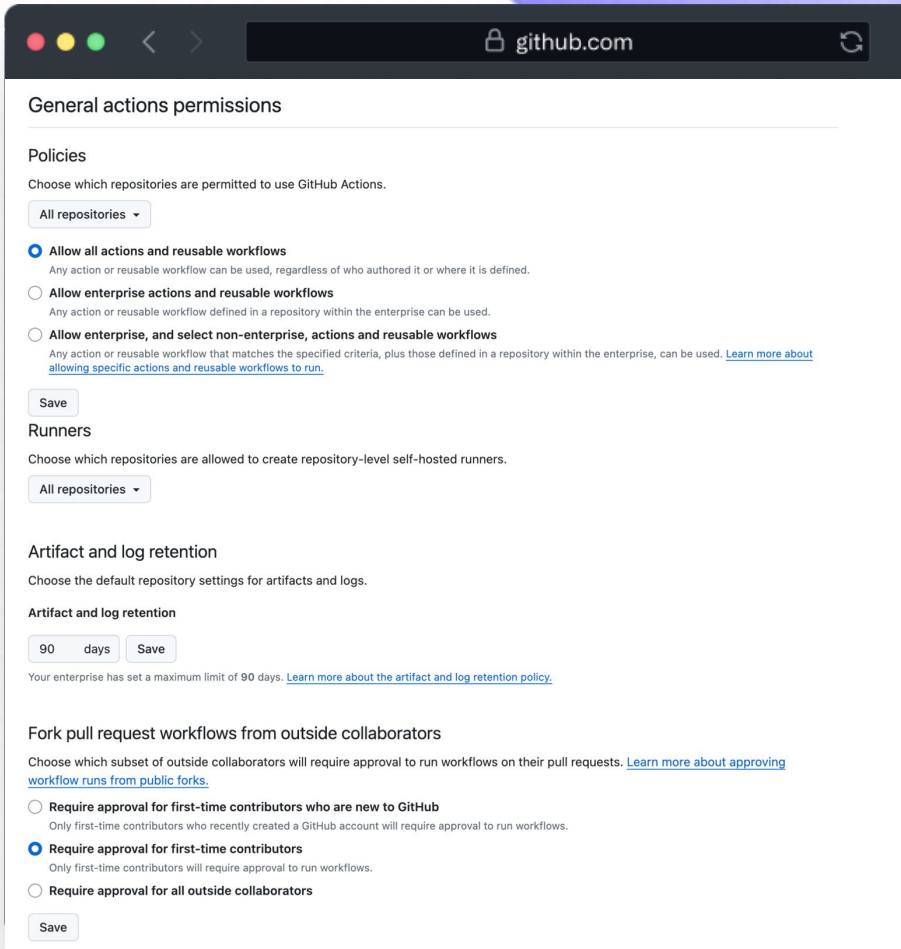
Demo

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



Sharing Workflows

Reusing Workflows

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

Centralized Secrets and Variables

Manage secrets and variables at the organization level

Centralized Management

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

Starter Workflows

Organizations can create starter workflows that act as templates

Self-Hosted Runners

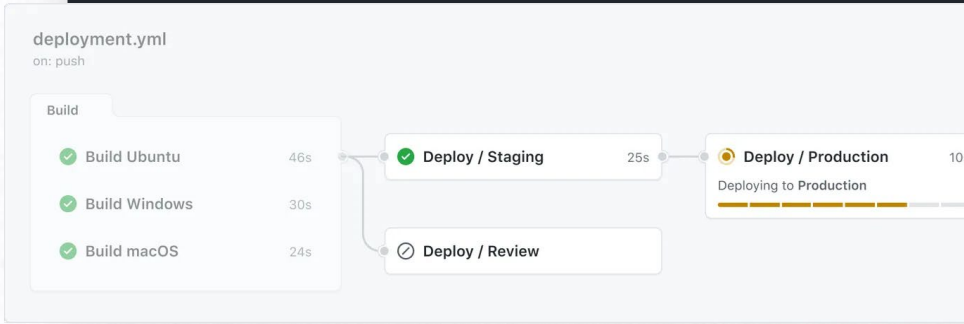
Organizations can group runners and control repository access via policies

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

github.com

```yaml
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/labeler@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

```yaml
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"
```

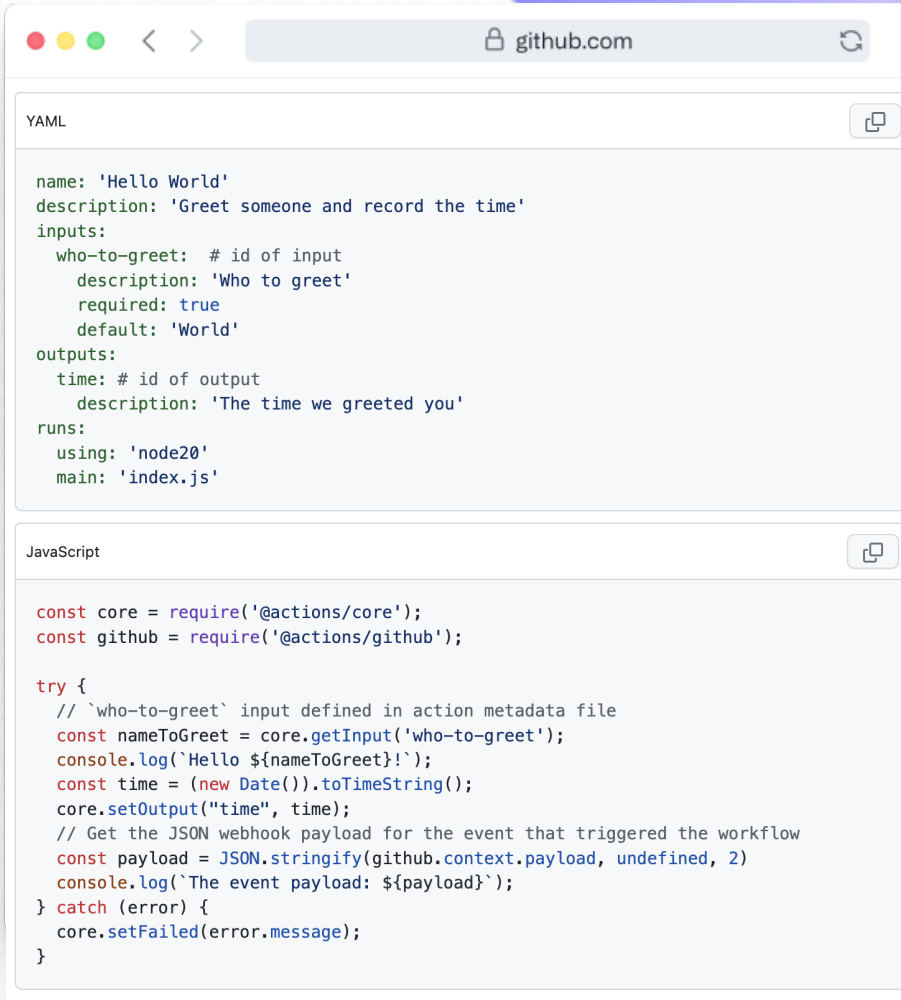
JavaScript Action

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



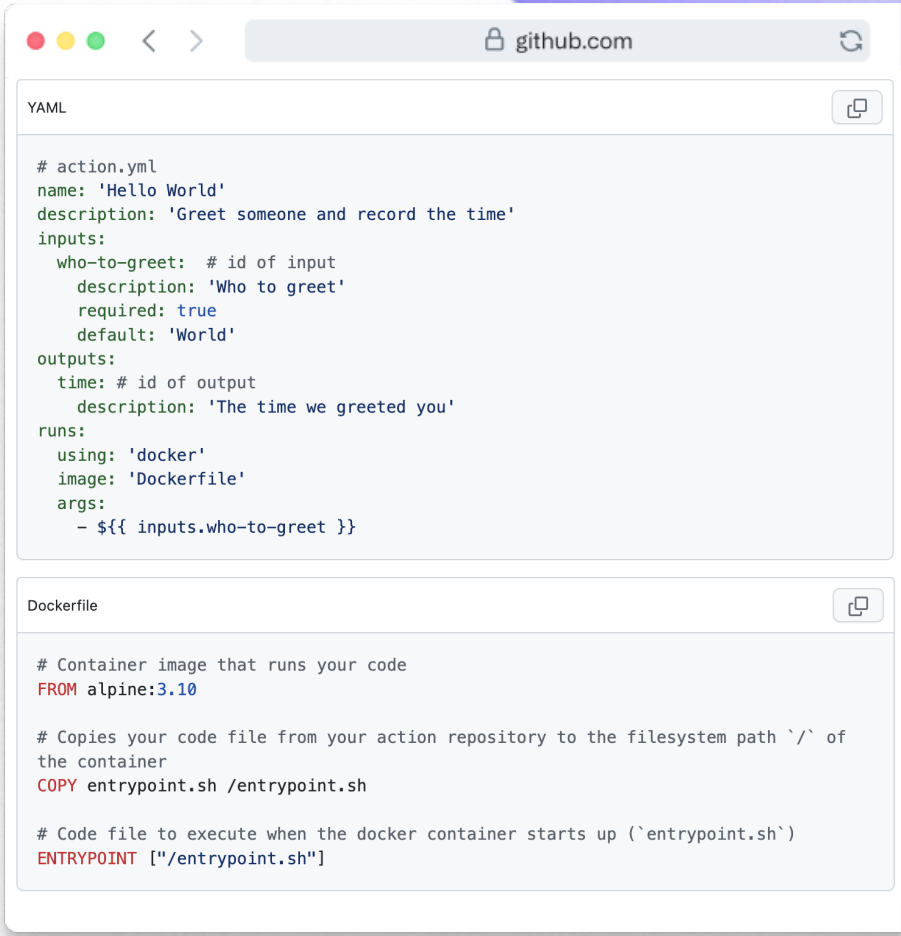
Docker Action

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

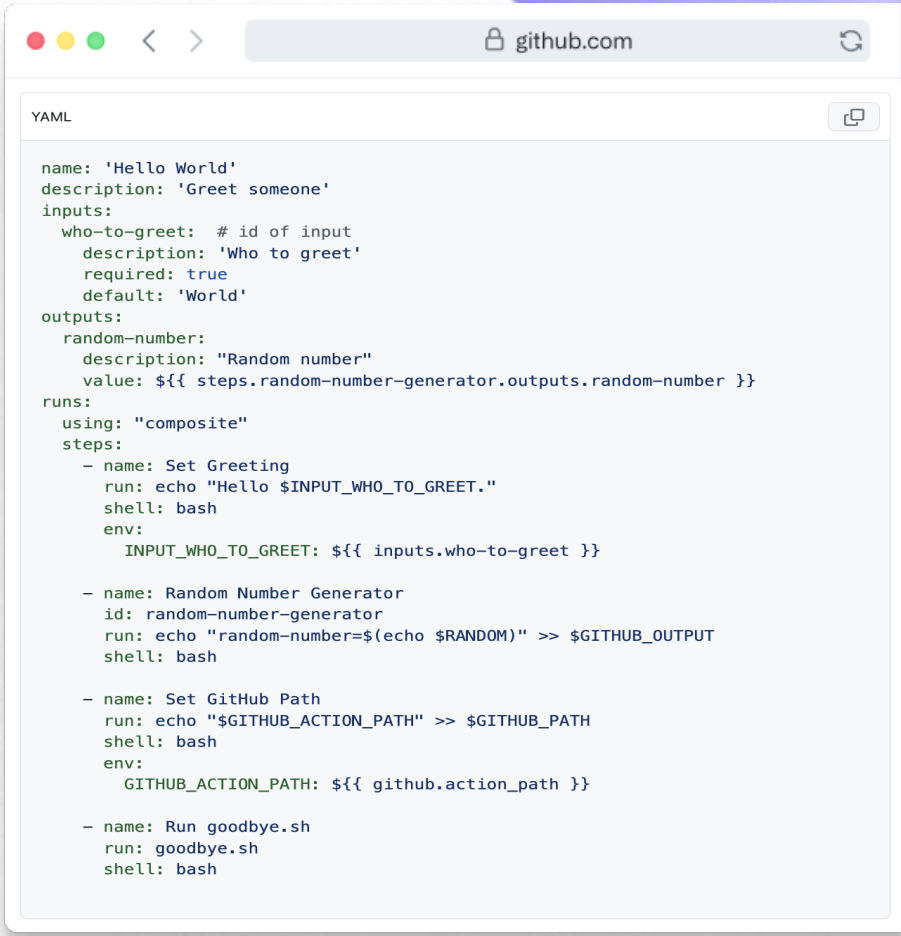


Composite Action

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`



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

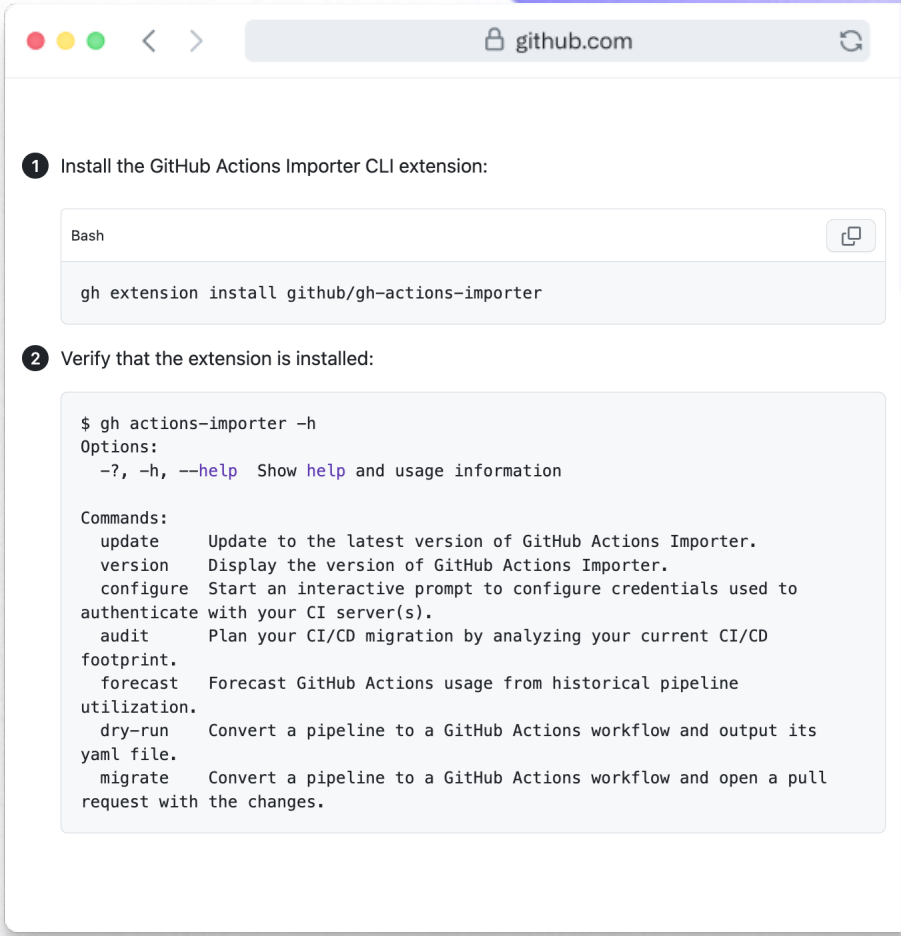
Faster onboarding

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.

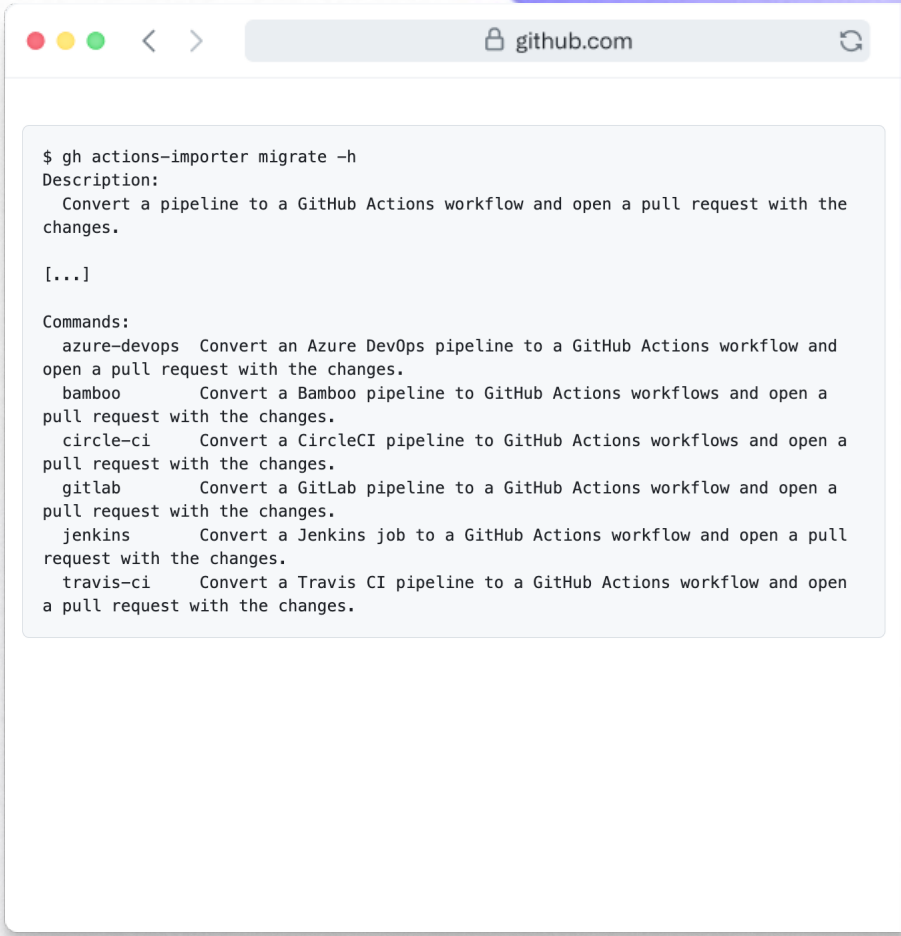
```
gh actions-importer audit -h
gh actions-importer forecast -h
gh actions-importer dry-run -h
gh actions-importer migrate -h
```



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



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

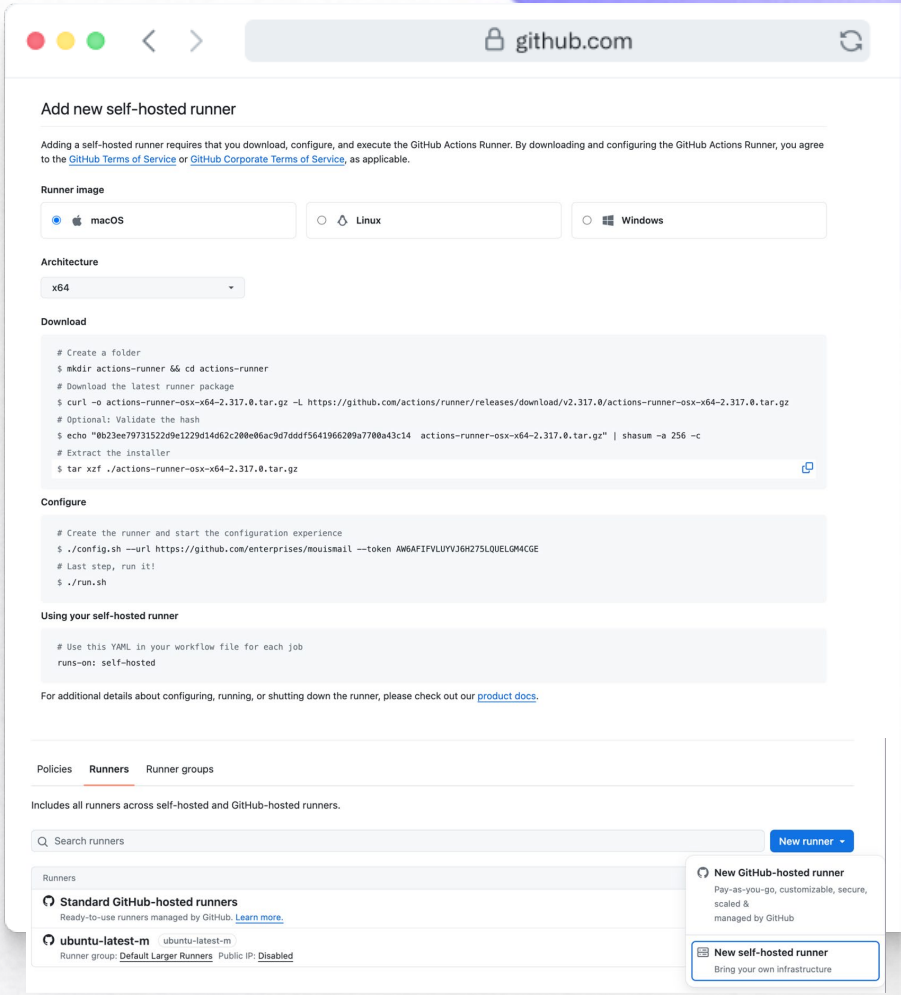
Self-hosted runners are machines that you manage and maintain to execute jobs from GitHub Actions workflows.

Provision the Runner

Install the Runner Application

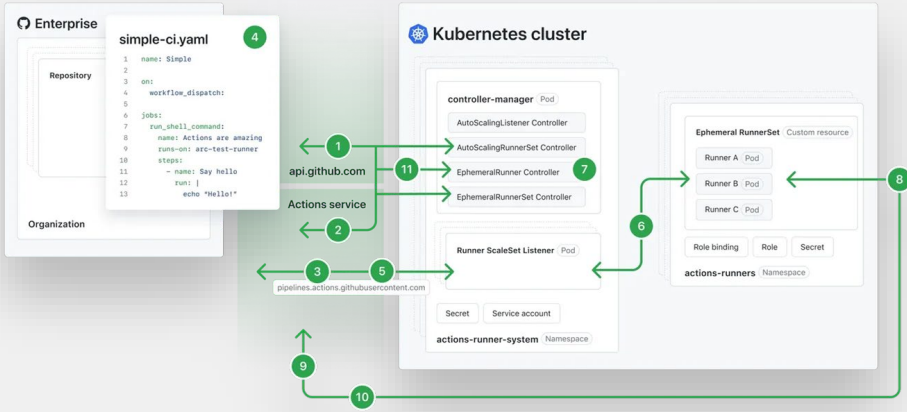
Configure the Runner

Manage and Maintain



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

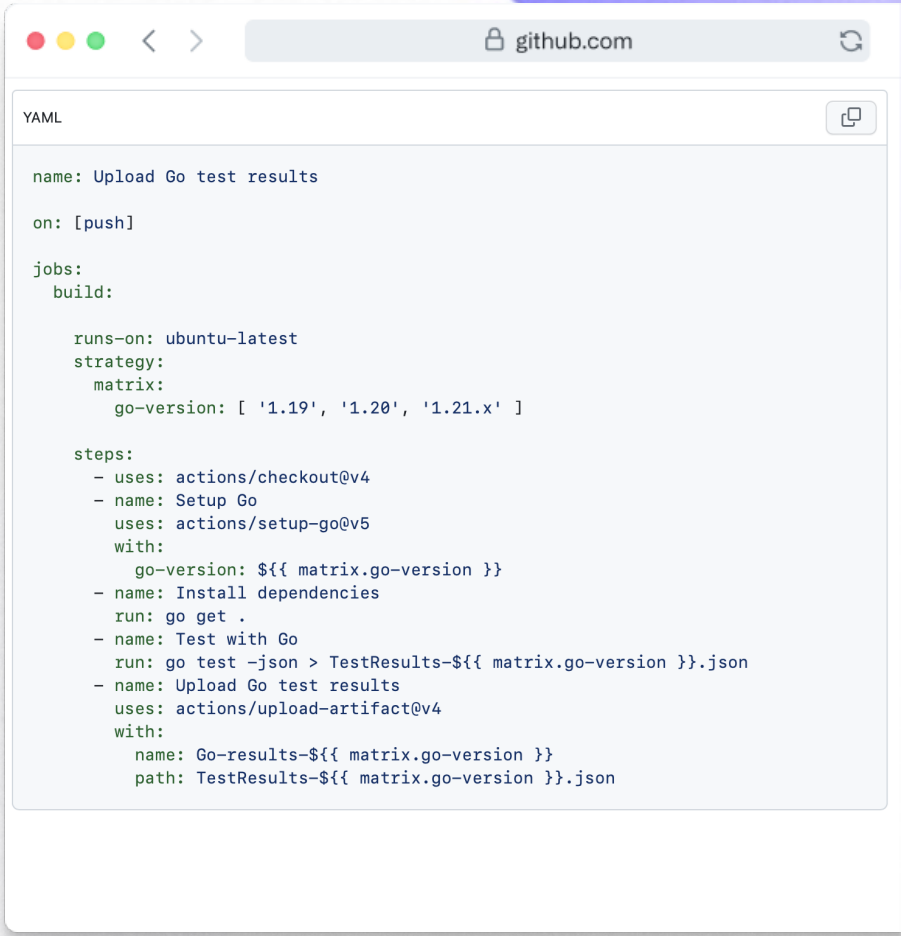
This guide shows you how to build, test, and publish a Go package

Specifying a Go version

Installing dependencies

Caching dependencies

Building and testing your code



Deploy

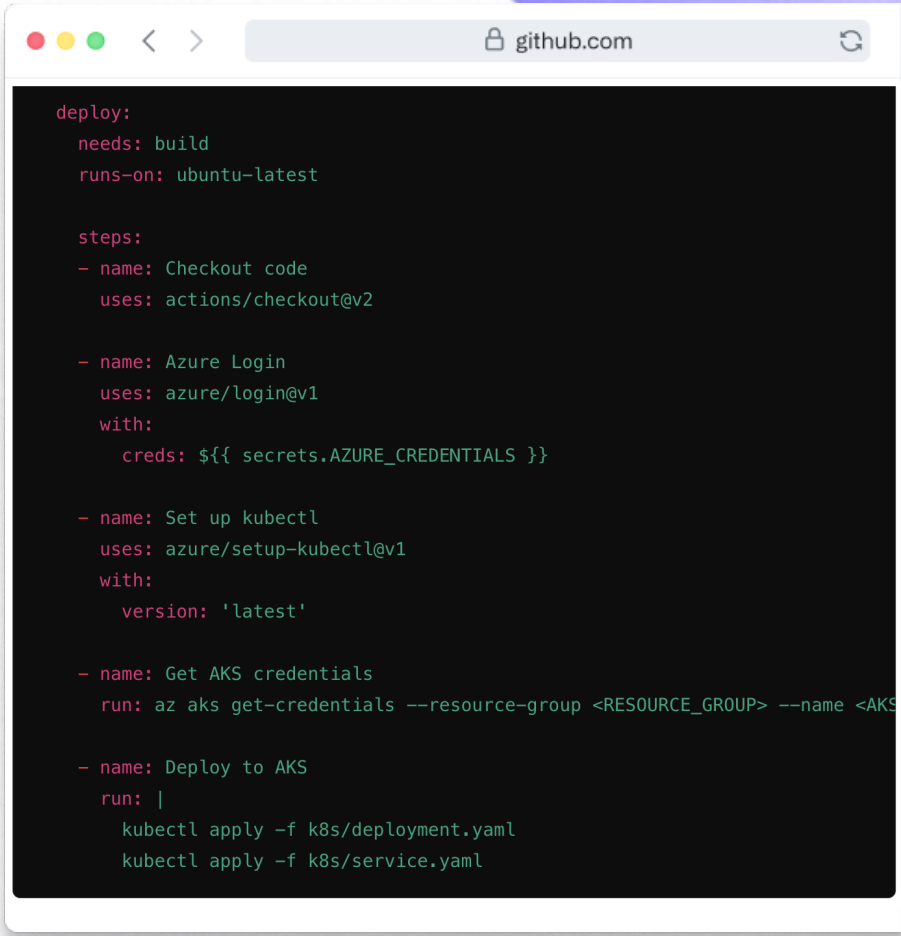
This guide explains how to use GitHub Actions to build and deploy a project to Azure Kubernetes Service

Azure Login

Build image on ACR

Configure deployment

Deploys application



What’s Next

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

Thanks