GitHub

Platform Integration 101

GitHub Partner Engineering

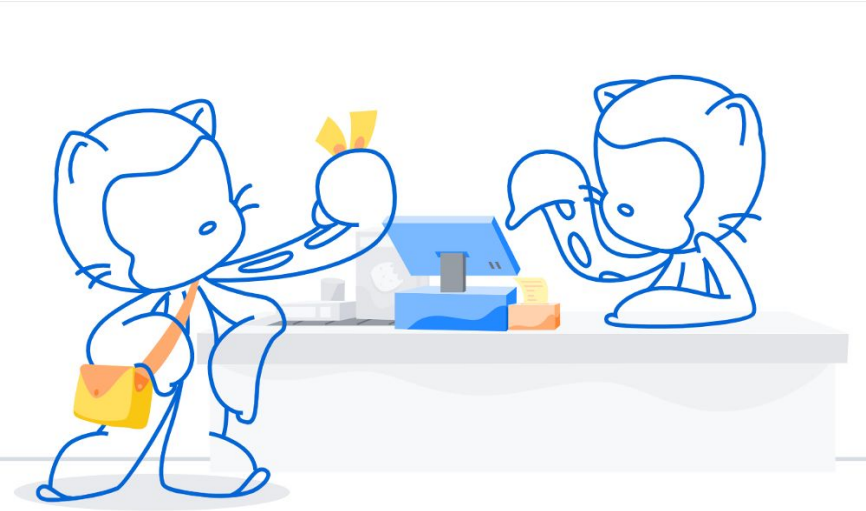
Agenda

- 👋 GitHub platform overview
- 🤖 GitHub Apps
- GitHub Actions
- 🚀 Summary

👋 GitHub platform overview

GitHub platform

- 60+ million developer reach
- Thousands of integrations in GitHub Marketplace
- Robust REST and GraphQL APIs
- Reliable webhook delivery
- App authentication model
- Built-in workflow automation tool called GitHub Actions
- Rich ecosystem of integrators and tooling
- Programs like secret scanning



Common touchpoints

- CI/CD flow (linting, parsing, scanning, commenting)
  - Checks API
  - Pull Requests API
  - Deployments API
  - Releases API
  - Git Data API
- Issue flow (conversation, utility, commenting)
  - Issues API
  - Reactions API
- Project Management flow
  - Repos API
  - Projects API
  - Organizations API
  - Teams API
- Security flow
  - Secret scanning and code scanning

GitHub Apps

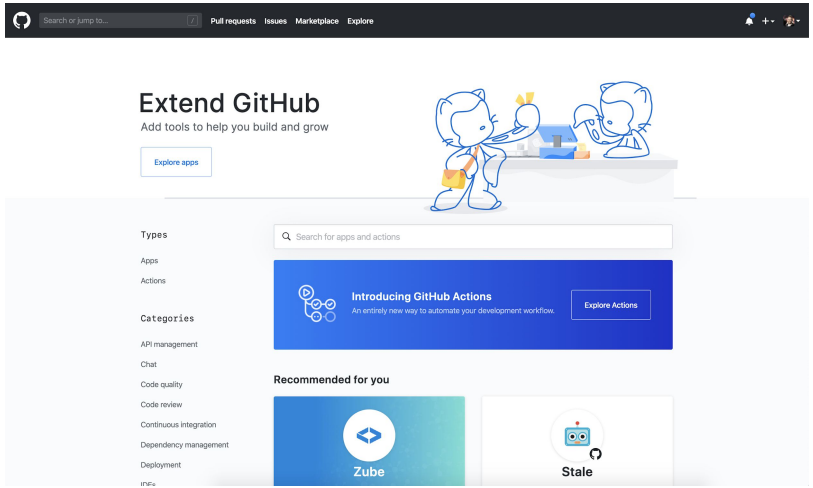
Introducing GitHub Apps

GitHub Apps are a tool to build comprehensive integrations with GitHub:

- First class actors on GitHub -- operating independently of any user identity
- Offer fine-grained permissions
- Installed on a user's or organization's repos
- Replace and offer many advantages over OAuth apps
- Come with built-in webhooks
- Work on GitHub.com and GitHub Enterprise Server
- Compatible with web technologies and standards, such as HTTP-based APIs and OAuth-like flows
- Rich open source tooling and libraries available, eg. octokit

Advantages for customer

- **Confidence** in granting third parties access to their assets in GitHub due to fine-grained and repo-centric permissions model
- **Convenience** through user-friendly (un)installation flow



Advantages for integrator

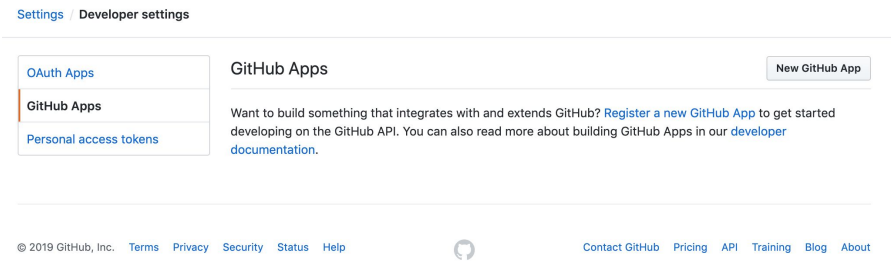
- Can **decouple** integration from GitHub user identities due to first class actor model of GitHub Apps.
- Can take advantage of **dedicated, scalable** **rate limits**, as opposed to the shared rate limit model offered by OAuth apps.
- Can utilize **modern GitHub APIs** like Checks and Content Attachments

Creating your first GitHub App

Option 1: Manual creation

1) Navigate to GitHub Apps from your GitHub Developer Settings
2) Register a New GitHub App, setting up URLs, permissions, and events
3) Download the private key and App ID and start coding!

For more, see here.



Option 2: Using Probot

Demo

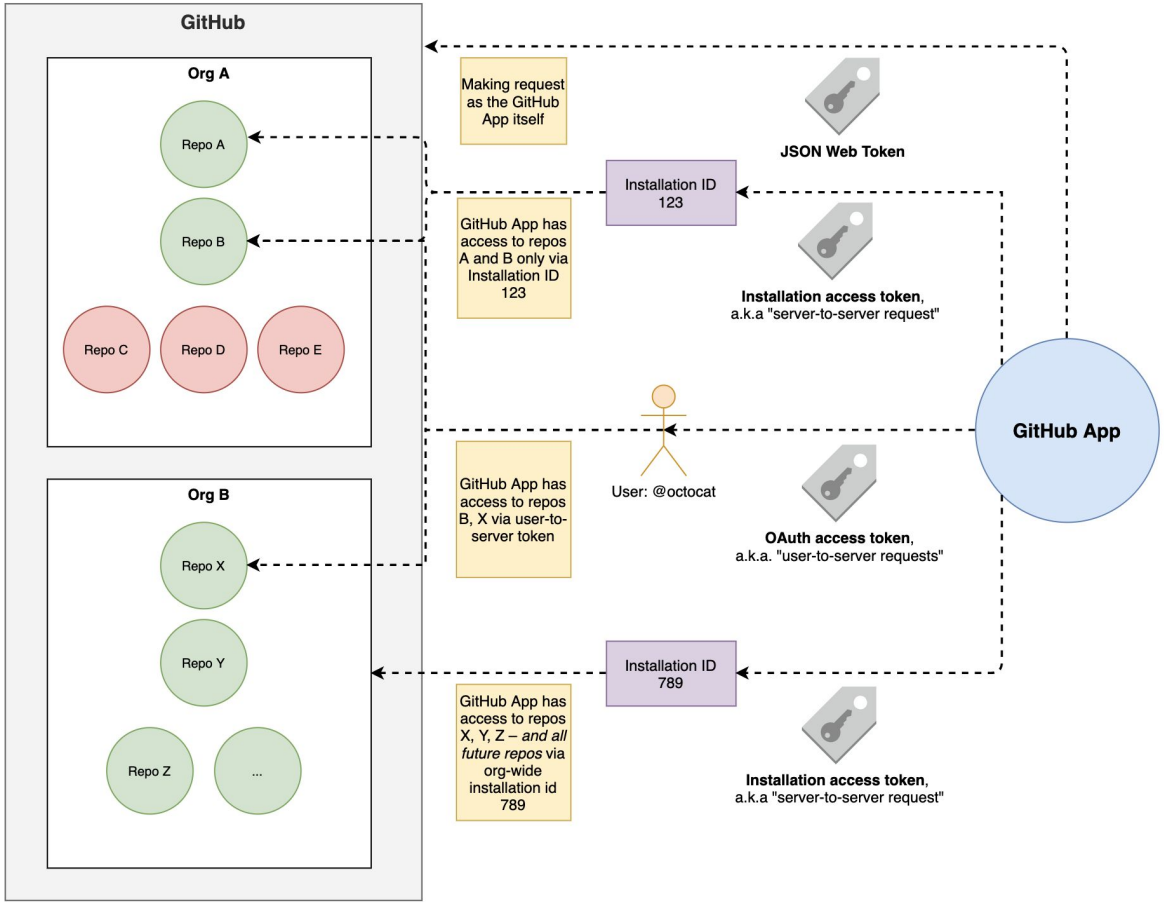
Authentication overview

Authentication Scheme	Also Known As	Description	How to Get It	Available Endpoints	Examples
JSON Web Token	JWT (pronounced “jot”)	Authenticates as the GitHub App	GitHub docs, Octokit	List	Fetching application installation details or exchanging the **JWT** for an **installation access token**.
Installation access token	Server-to-server requests	Authenticates as a specific installation of the GitHub App	GitHub docs, Octokit	List	Opening an issue or providing feedback on a pull request
OAuth access token	User-to-server requests	Authenticates as a user of the GitHub App	GitHub docs	List	Authenticating as a user when a GitHub App needs to verify a user’s identity or act on a user’s behalf

Authentication at a glance

Deciding which authentication type to use comes down to:

- What resource do I need to access?
- Who do I need to access it as?



Server-to-server requests

Server-to-server requests are those made from the perspective of an *installation* and are authenticated by **installation access tokens.**

Using your **JWT**, generate an **installation access token** via:

```
curl -i -X POST \
      -H "Authorization: Bearer YOUR JWT" \
      -H "Accept: application/vnd.github.machine-man-preview+json" \
      https://api.github.com/app/installations/:installation_id/access_tokens
```

As a security measure, these tokens expire after 1 hour. They can be used like:

```
curl -i \
      -H "Authorization: token YOUR INSTALLATION ACCESS TOKEN" \
      -H "Accept: application/vnd.github.machine-man-preview+json" \
      https://api.github.com/installation/repositories
```

User-to-server requests

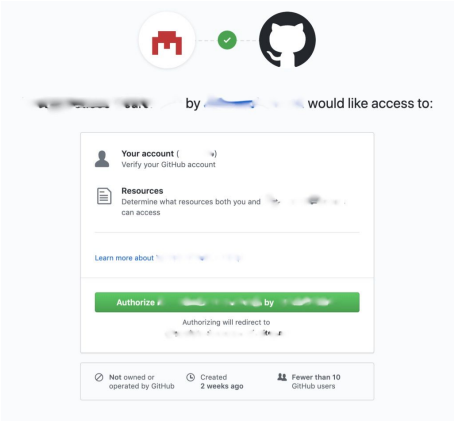
User-to-server requests act as a *user who has authorized your GitHub App* and are authenticated using an **OAuth access token**.

First, users authorize your GitHub App [via OAuth]() and receive a `code`:

Then, your GitHub App trades the `code`, `client_id` and `client_secret` for an **OAuth access token** to be used like:

```
curl -H "Authorization: token OAUTH-TOKEN" https://api.github.com/user
```

Unlike typical OAuth, the scope is determined by the GitHub App.



Onboarding new users

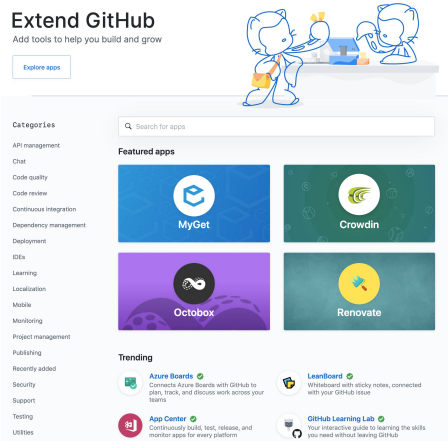
Optimal flow (Demo)

1. Optional -- User purchases app on [GitHub Marketplace]

2. User installs app on repositories and authorizes the app

3. GitHub redirects to app's registered *callback URL*

4. App exchanges OAuth code for access token



Notable APIs for GitHub Apps

- GitHub App information
  - Get the authenticated GitHub App (JWT)
- Identify installation information
  - List installations (JWT)
  - Get an organization installation (JWT)
  - Get a user installation (JWT)
- Token creation / revocation
  - Create a new installation token (JWT)
  - Revoke an installation token (installation access token)
- Identify installation resources
  - List repositories (installation access token)
- Identify user-accessible resources
  - List installations for a user (user-to-server OAuth access token)
  - List repositories accessible to the user for an installation (user-to-server OAuth access token)

GitHub Apps best practices

✅ **Do:**

- Cache and re-use installation tokens
- Use webhooks for real-time data
- Throttle requests to stay within rate limits
- Consider if REST or GraphQL APIs (or both) are best for your use case
- Use conditional requests wherever possible
- Subscribe to this RSS feed for Platform updates
- Include a descriptive User-Agent header
- Save the `X-GitHub-Request-Id` response header value, especially for error responses
- Follow other best practices listed here

❌ **Don't:**

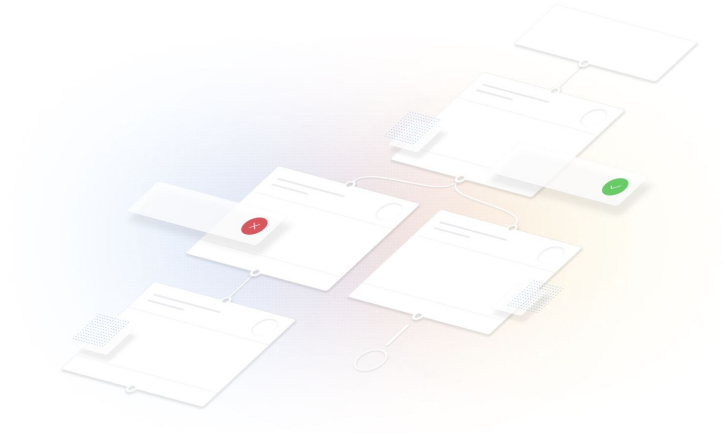
- Depend on concurrent requests, this can trigger secondary rate limits
- Poll, use webhooks where possible

GitHub Actions

Introducing GitHub Actions

GitHub Actions makes it easy to automate all your software workflows, now with world-class CI/CD.

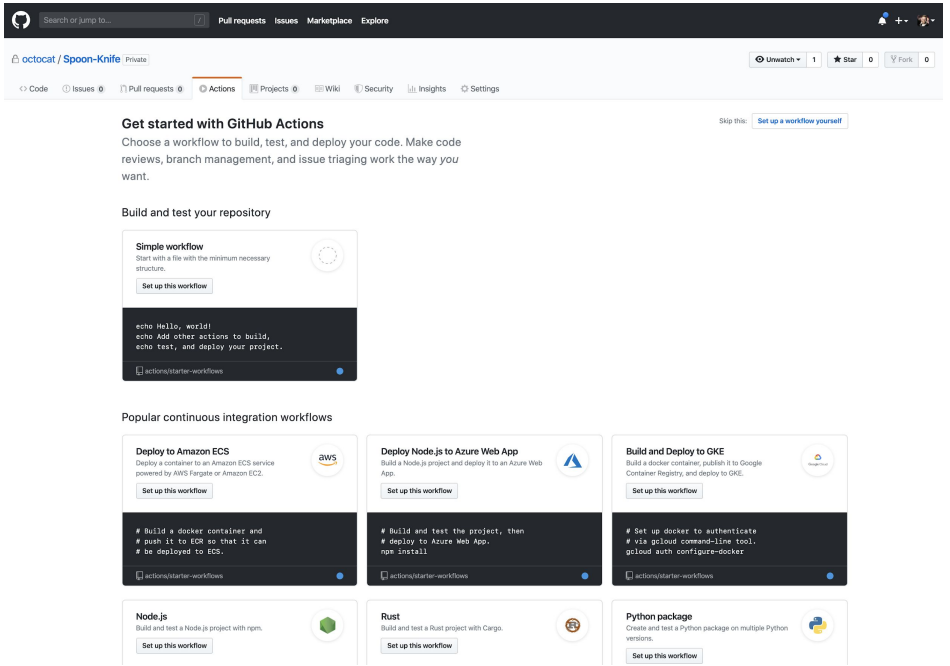
- Built in CI/CD
- Linux, Mac, Windows, and containers
- Matrix builds
- Easy to write, easy to share
- Streaming, searchable, linkable logs
- Built-in secret store
- Artifact caching
- Self hosted runners
- Event-driven or schedule-driven
- GitHub Enterprise Server support soon



Getting started

1. Head to the Actions tab on any of your repositories.

2. Set up a workflow using one or more actions, triggered upon event or on schedule. ✨



Workflow

A configurable automated process that you can set up in your repository.

For example:

- Organizational: Welcoming new contributors
- Legal: Ensuring license uniformity
- Application: Testing across multiple operating systems

```yaml
# .github/workflows/build.yml
name: Node CI

on: [push]

jobs:
  build:

    runs-on: ubuntu-latest

    strategy:
      matrix:
        node-version: [8.x, 10.x, 12.x]

    steps:
    - uses: actions/checkout@v1
    - name: Use Node.js ${{ matrix.node-version }}
      uses: actions/setup-node@v1
      with:
        node-version: ${{ matrix.node-version }}
    - run: npm install
    - run: npm run build --if-present
    - run: npm test
      env:
        CI: true
```

Event

Workflows are triggered on events.

For example:

- push, pull_request, public, etc.
- schedule
- workflow_dispatch (manual trigger)
- repository_dispatch (outside systems)

```yaml
# .github/workflows/weekly-radar
name: Weekly Radar

on:
  schedule:
  - cron: 0 12 * * 1

jobs:

  weekly_radar:
    name: Weekly Radar
    runs-on: ubuntu-latest
    steps:

    - name: weekly-radar
      uses: imjohnbo/weekly-radar@master
      with:
        assignees: "teammate1 teammate2"
        pinned: true
      env:
        GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
```

Action

Individual unit of work that you combine as steps to create a job in a workflow.

For example:

- actions/checkout
- actions/cache
- actions/javascript-action
- Lots more on GitHub Marketplace

```yaml
# action.yml
name: 'Wait'
description: 'Wait a designated number of milliseconds'
inputs:
  milliseconds:  # id of input
    description: 'number of milliseconds to wait'
    required: true
    default: '1000'
outputs:
  time: # output will be available to future steps
    description: 'The message to output'
runs:
  using: 'node12'
  main: 'index.js'
```

```javascript
// index.js
const core = require('@actions/core');
const wait = require('./wait');

async function run() {
  try {
    const ms = core.getInput('milliseconds');
    console.log(`Waiting ${ms} milliseconds ...`)

    core.debug((new Date()).toTimeString())
    wait(parseInt(ms));
    core.debug((new Date()).toTimeString())

    core.setOutput('time', new Date().toTimeString());
  }
  catch (error) {
    core.setFailed(error.message);
  }
}

run()
```

Actions API

Actions is [backed by a RESTful API](), allowing programmatic access to workflows, artifacts, secrets, and even self-hosted runners.

For example:

- [List artifacts for a repository]()
- [Re-run a workflow]()
- [Create or update a secret]()
- [List self-hosted runners for a repository]()
- More information in [this blog]()
- [Example implementation]() in Marketplace
- Full capabilities described in [the docs]()

```
# Store/update a secret for a repository
curl --request PUT \
  --url https://api.github.com/repos/:owner/:repo/actions/secrets/:secret \
  --header 'Accept: application/vnd.github.jane-hopper-preview+json' \
  --header 'Authorization: Bearer bf1d33cec63a87ff70d6f7195ab292ef4219a70f' \
  --header 'Content-type: application/json' \
  --data '{
  "encrypted_value":
"I2b/vYLiGJcrZvJZ1Vdeqef9UDHRN2+MHIFUuDmc8iajzkZEJvSV9lsk99uXqHLWm4tXGoohce22XRdhoW+B",
  "key_id": "0123456789012345678 9"
}'
```

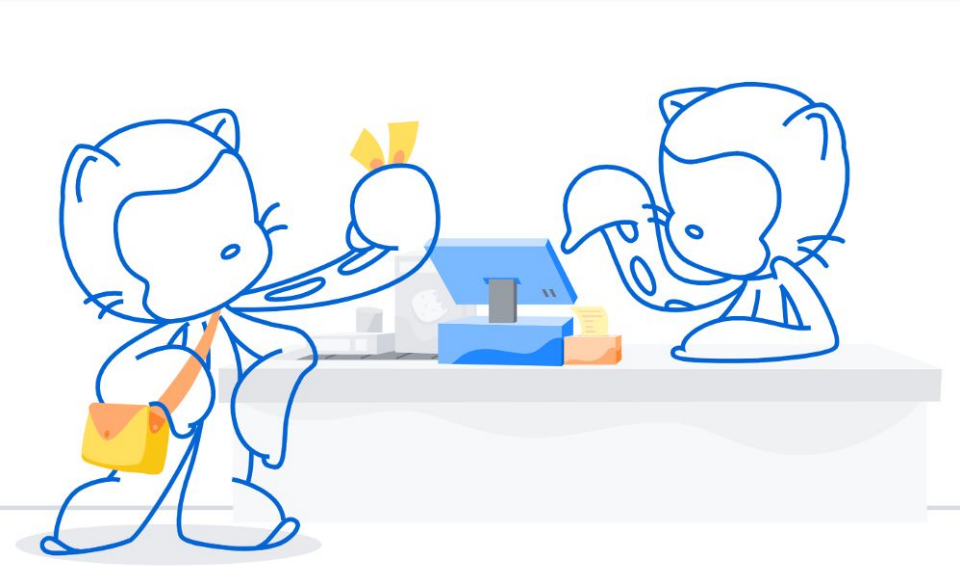
```
# Re-run a workflow
curl --request POST \
  --url https://api.github.com/repos/:owner/:repo/actions/runs/:run_id/rerun
\
  --header 'Authorization: Bearer bf1d33cec63a87ff70d6f7195ab292ef4219a70f'
```

Community

GitHub Actions is powered by an open ecosystem and community contributions.

For example:

- Workflows: actions/starter-workflows
- Actions: github.com/actions
- Tooling: actions/toolkit



GitHub Actions best practices

✅ **Do:**

- Prefer JavaScript to container
- Prefer chainable to monolithic
- Documentation, examples, blog posts, releases, `LICENSE`
- GitHub Marketplace for discoverability
- Use open source tooling (eg. `@vercel/ncc`, `actions/toolkit`)
- Use inputs and outputs

❌ **Don't:**

- Produce undocumented side effects
- Waste users' runner minutes (they'll notice)

GitHub Apps vs. GitHub Actions

Apps

- Your integration requires user interaction
- Your integration needs to handle state
- Your integration acts across multiple repos, or at the organization level
- Your integration is available to the public but no part of the code is public
- You are comfortable hosting the app yourself
- You need permissions that are outside the set provided by Actions
- You need events that are outside the set provided by Actions

Actions

- Your integration is essentially “headless”, i.e. it does not require user interaction, or uses GitHub.com for its user interface
- Your integration does not need to persist data in a database
- Your integration wraps an existing CLI, or API
- You are comfortable with your action code being publicly visible
- You would prefer GitHub to run your integration

GitHub hosted runners

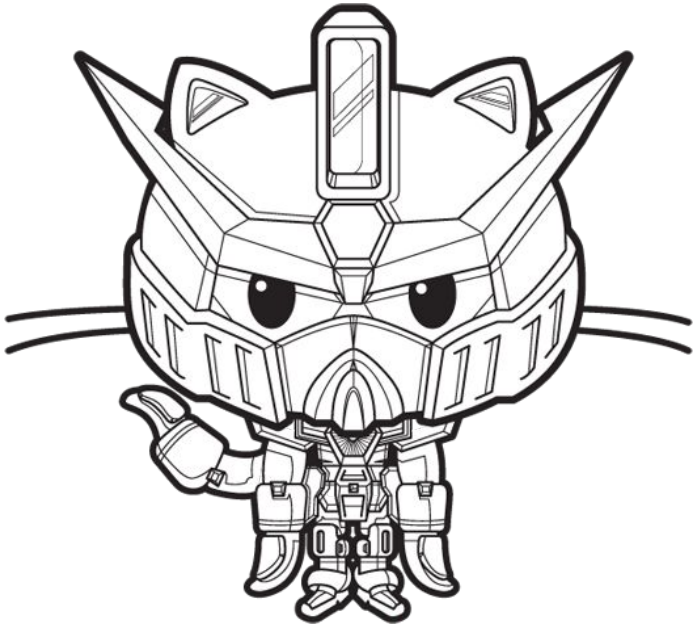
- “Just Works” solution for maximum ease
- Linux, Windows, macOS
- Compatible with public, internal, and private repos
- Ephemeral runner VMs in predictable environment
- Integrated billing and security model
- GitHub provides machine maintenance and upgrades

Self hosted runners

- Custom solution for maximum control
- Linux, Windows, macOS supported, container possible
- Recommended only for private repos
- Can utilize custom hardware and processor architectures (e.g. ARM) and operate in your network
- You assume responsibility for environment, security, billing, and management

Libraries and resources

- Developer Documentation
- GitHub REST and GraphQL APIs
- GitHub Apps
- GitHub Webhooks
- Octokit
- Probot
- Actions Documentation
- Example JavaScript action
- Example container action
- Actions toolkit
- GitHub Marketplace, filterable by Apps or Actions



🚀 Summary

GitHub Apps are more **FLEXIBLE**, and **POWERFUL**, but come with some overhead (mostly hosting the app) 💪

GitHub Actions are **SMALLER**, more **LIGHTWEIGHT**, and probably will **JUST WORK** for most integration needs ✨

GitHub APIs are available through either type of integration.