

Building and Securing CI/CD Pipelines for Embedded Systems using GitHub Actions

12/05/2026



SOFTWARE
LANGUAGES
LAB

This Workshop

Overview

PART 1:

SETTING UP A CI/CD PIPELINE
USING GITHUB ACTIONS

PART 2:

CI/CD PIPELINE SECURITY

This Workshop

Overview

PART 1:

SETTING UP A CI/CD PIPELINE
USING GITHUB ACTIONS



This Workshop

Overview

PART 1: **SETTING UP A CI/CD PIPELINE USING GITHUB ACTIONS**

GitHub Actions preliminaries

Setup of a simple CI/CD pipeline



Setting Up a CI/CD Pipeline

Continuous **I**ntegration / Continuous **D**elivery

Setting Up a CI/CD Pipeline

Continuous **I**ntegration / **C**ontinuous **D**elivery

“**shift left**”: detect (security) vulnerabilities early on

“**reproducibility**”: automated builds

Setting Up a CI/CD Pipeline

Continuous Integration / Continuous Delivery

“**shift left**”: detect (security) vulnerabilities early on

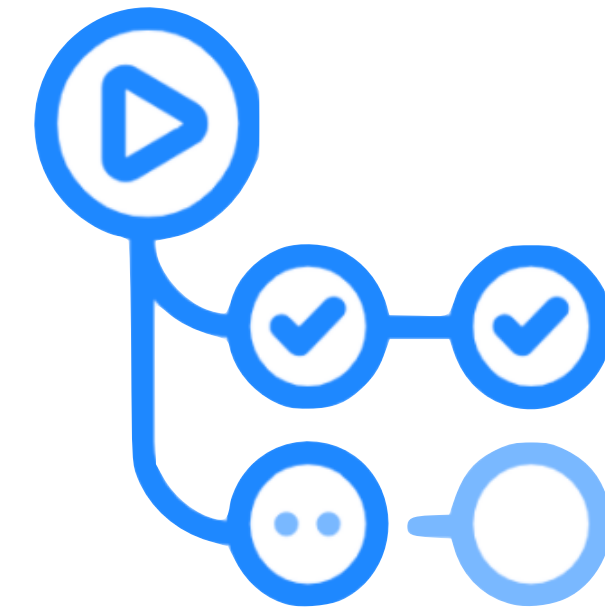
“**reproducibility**”: automated builds



Jenkins



CI/CD



GitHub Actions

Setting Up a CI/CD Pipeline

Continuous Integration / Continuous Delivery

“**shift left**”: detect (security) vulnerabilities early on

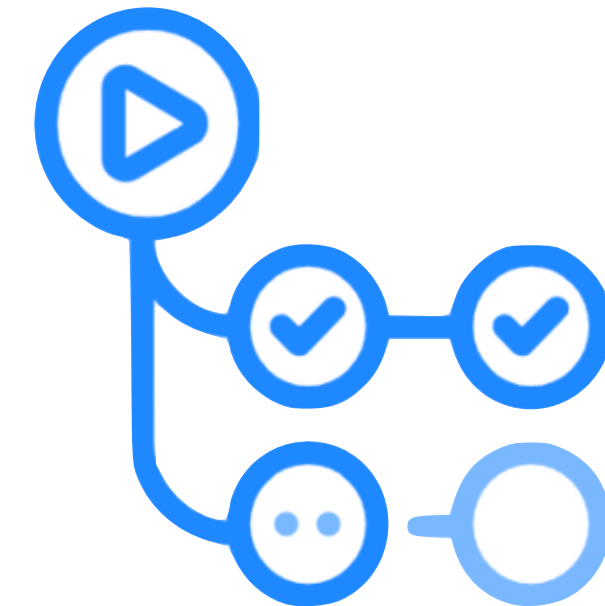
“**reproducibility**”: automated builds



Jenkins



CI/CD



GitHub Actions

Setting Up a CI/CD Pipeline

Continuous Integration / Continuous Delivery

“**shift left**”: detect (security) vulnerabilities early on

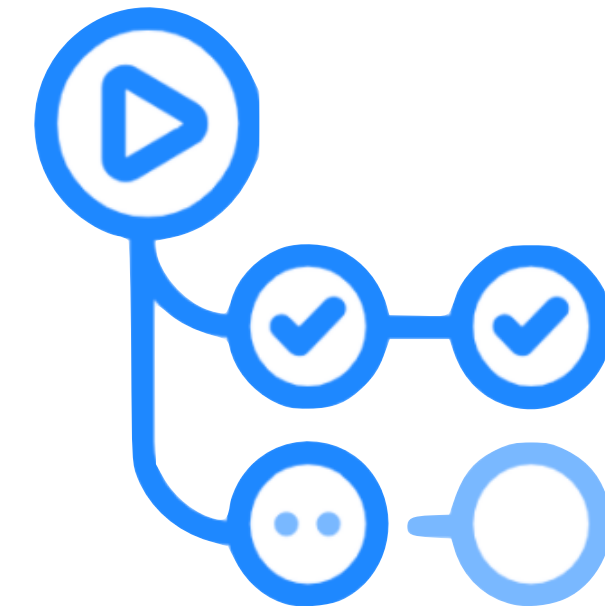
“**reproducibility**”: automated builds



Jenkins



CI/CD



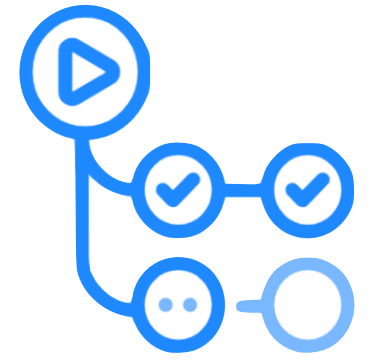
GitHub Actions

Migrating?

<https://docs.github.com/en/actions/tutorials/migrate-to-github-actions>



What's in a Pipeline?



GitHub Actions

Software Composition Analysis (SCA)

Dependency Scanning

SBOM Generation

Static Application Security Testing (SAST)

Dynamic Application Security Testing (DAST) / Fuzzing

Unit & Integration Testing

IaC Security Scanning

Build

Firmware Analysis

Compliance Checking

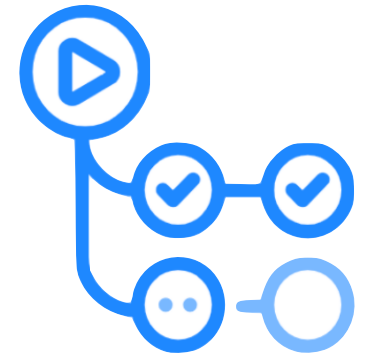
Secure Deployment

Secrets Scanning

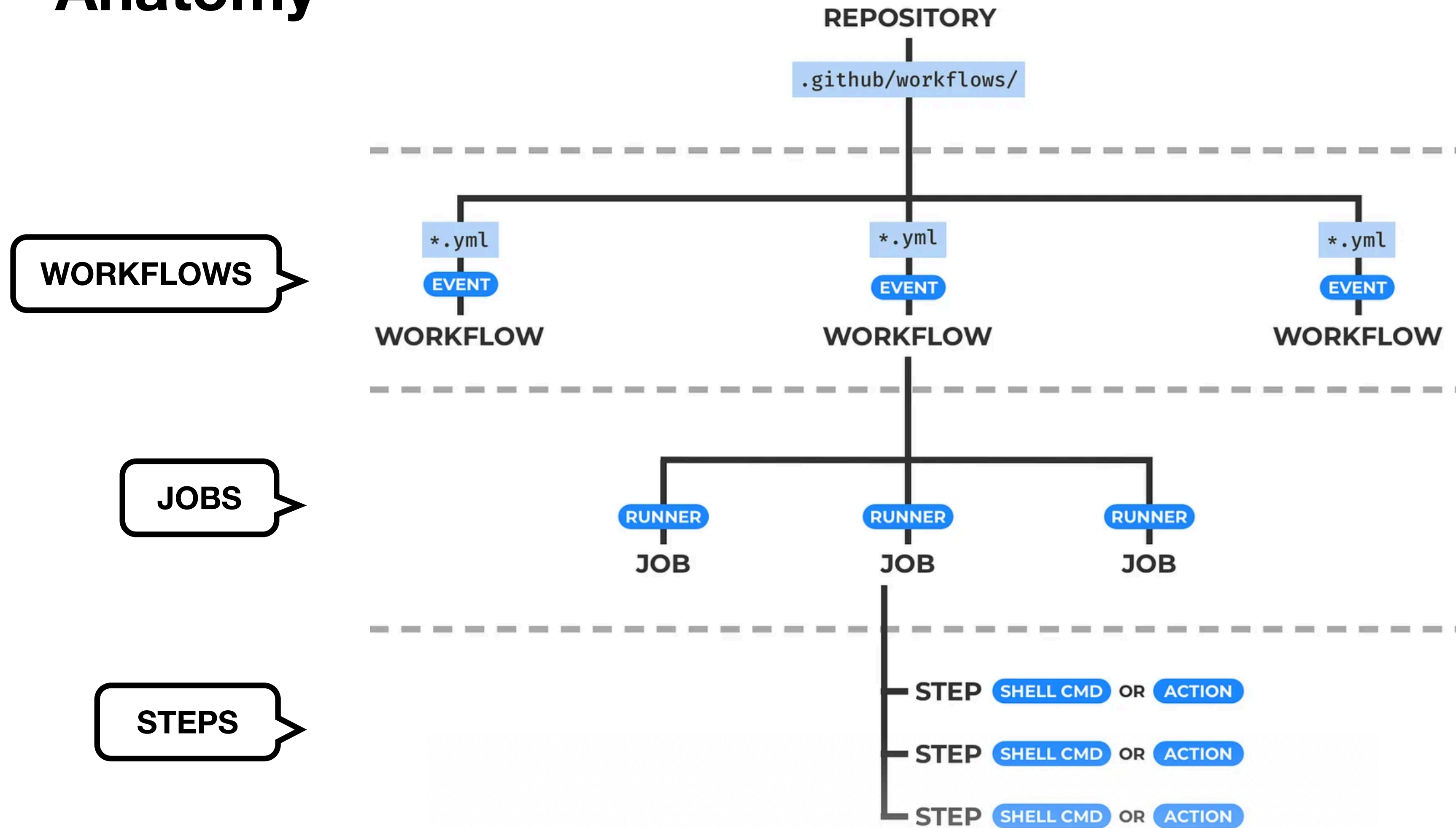
Artifact/Firmware Signing

GitHub Actions: A Primer

Anatomy

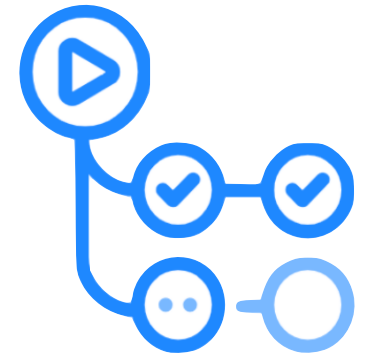


GitHub Actions

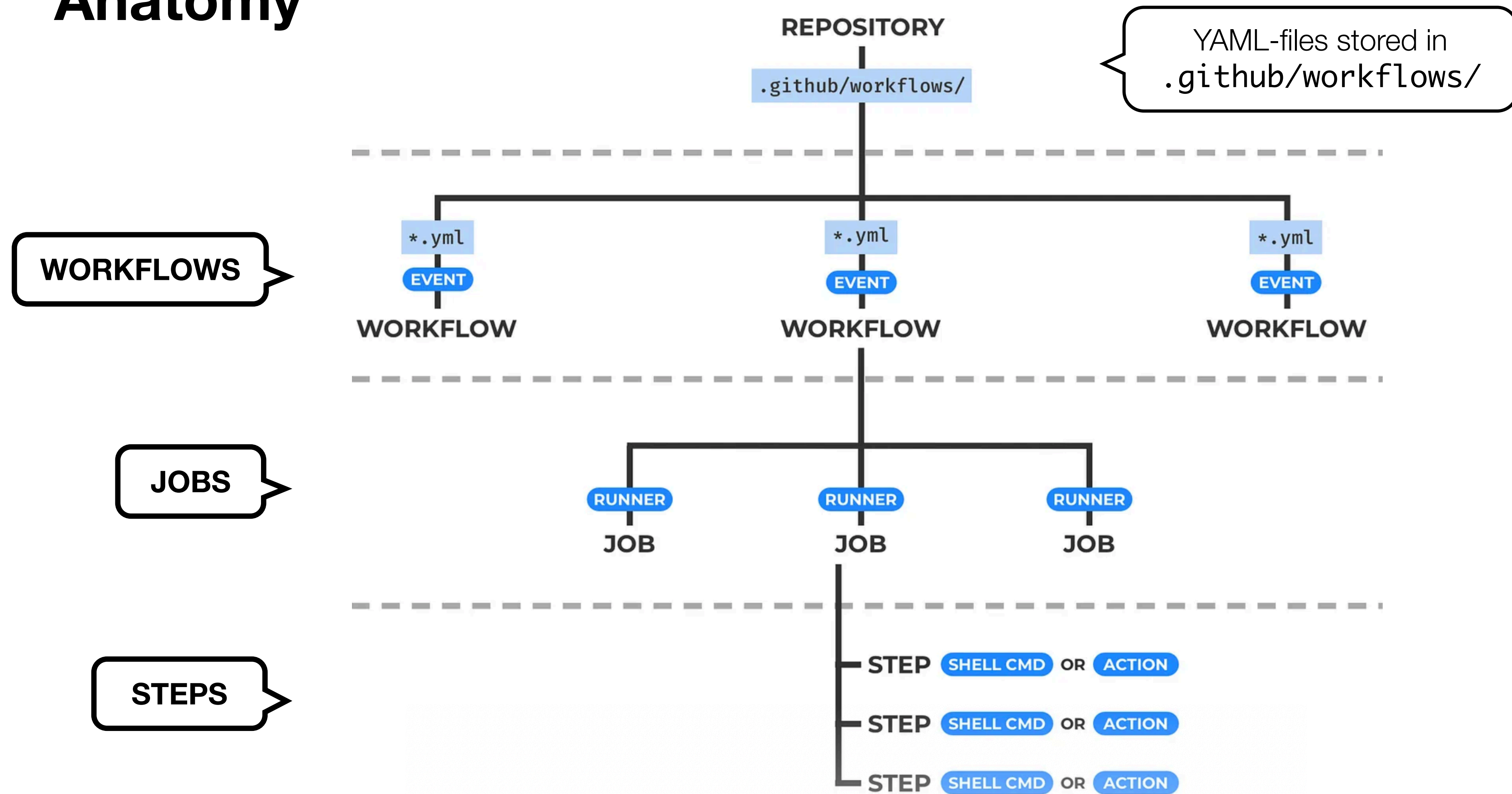


GitHub Actions: A Primer

Anatomy

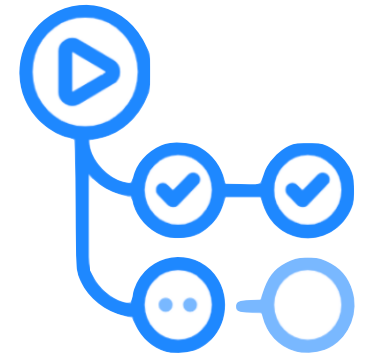


GitHub Actions

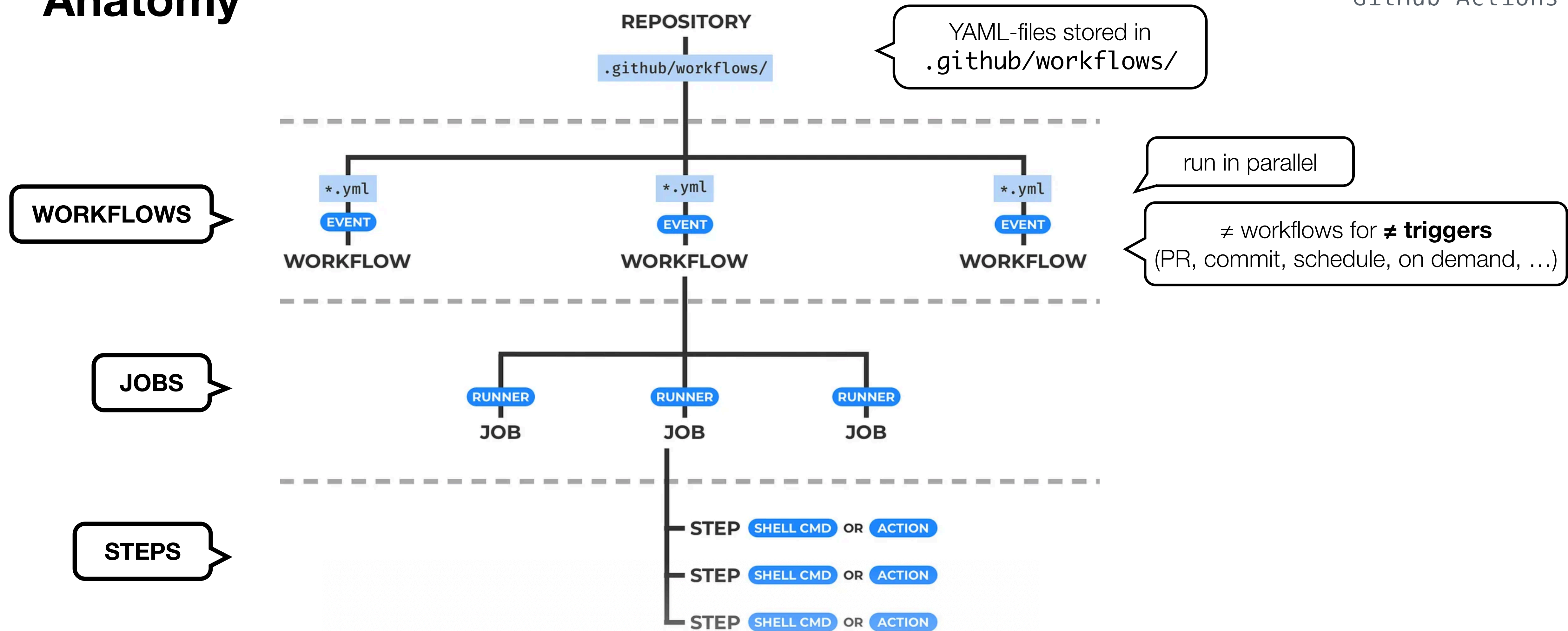


GitHub Actions: A Primer

Anatomy

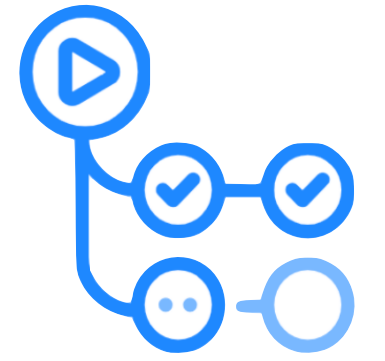


GitHub Actions

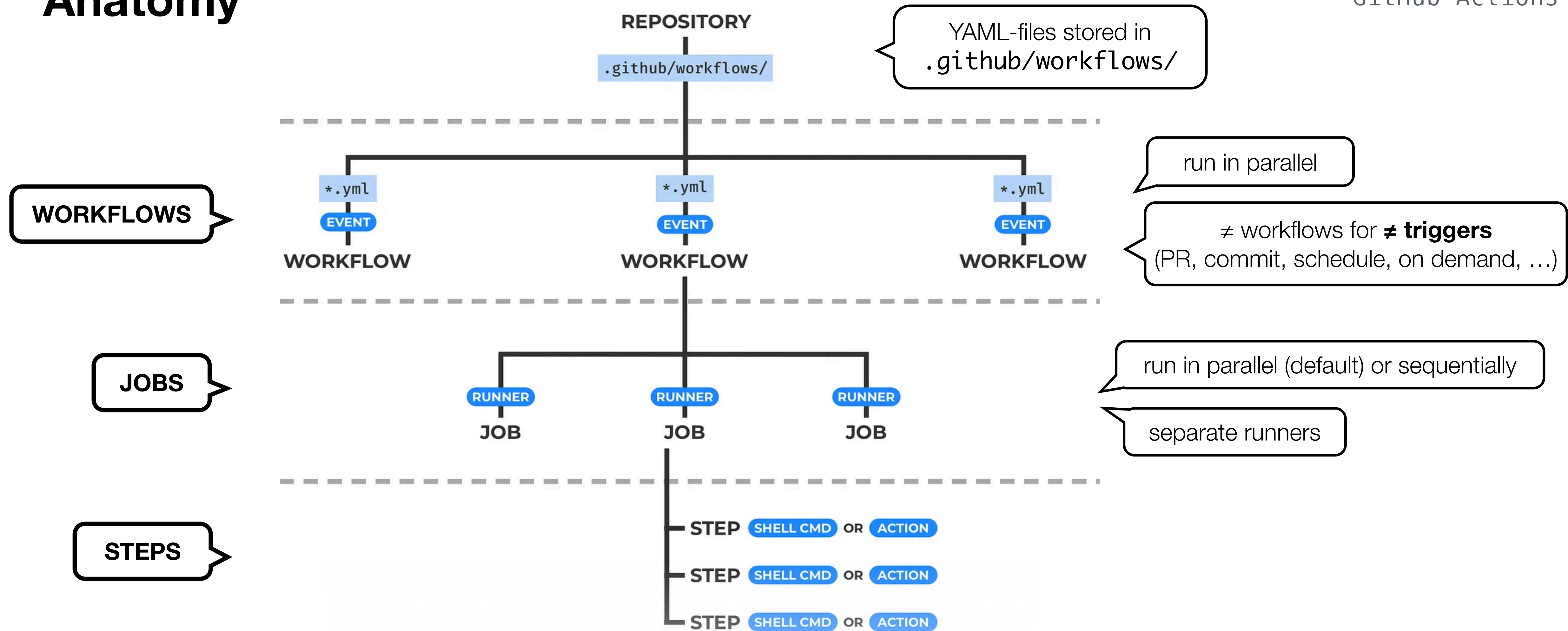


GitHub Actions: A Primer

Anatomy

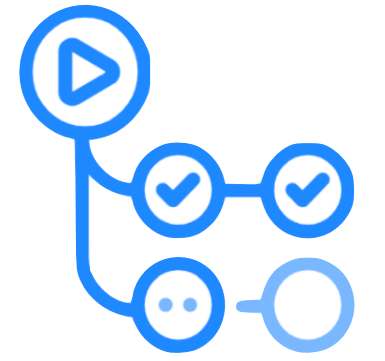


GitHub Actions

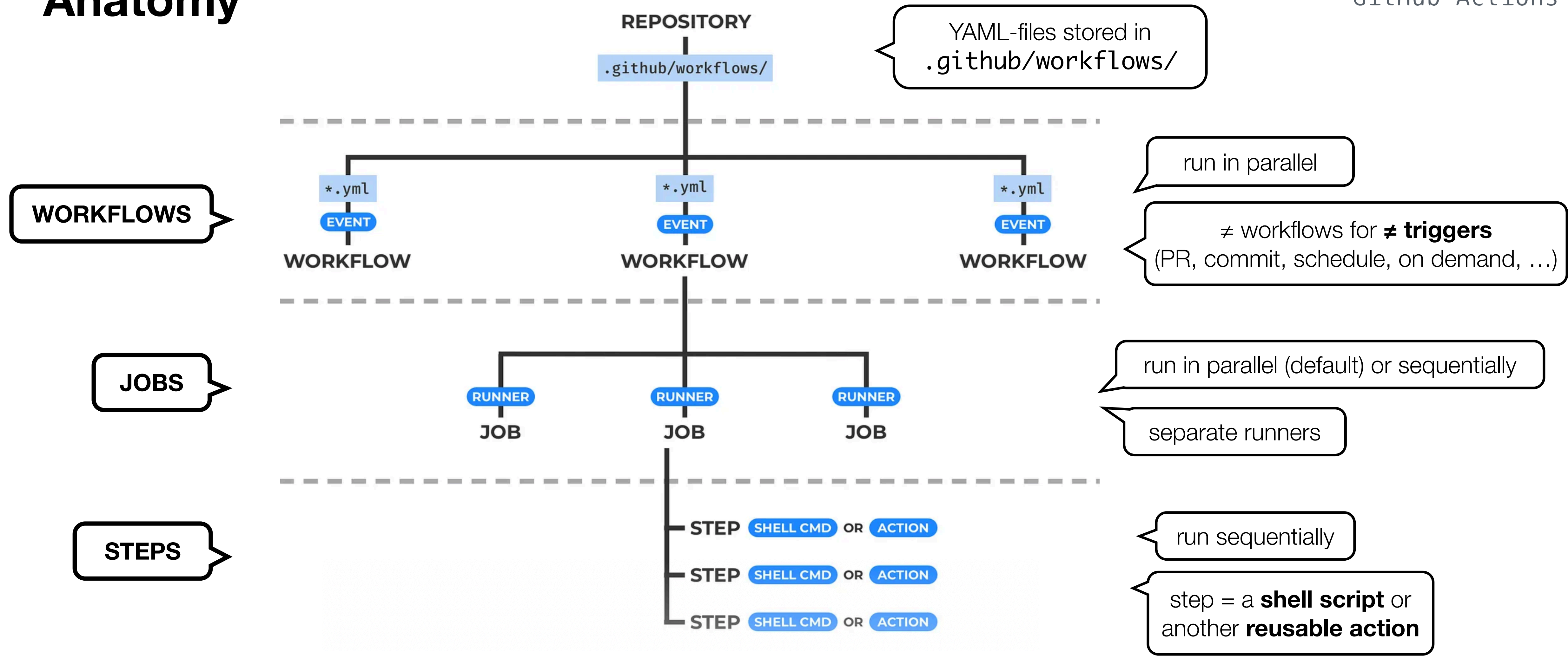


GitHub Actions: A Primer

Anatomy

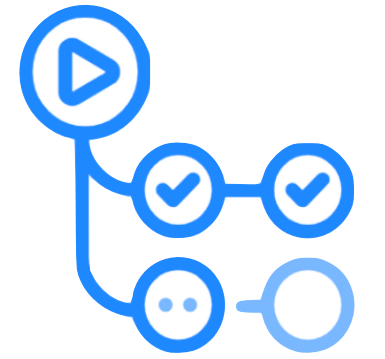


GitHub Actions



GitHub Actions: A Primer

Example Repository



GitHub Actions

▼  .github/workflows

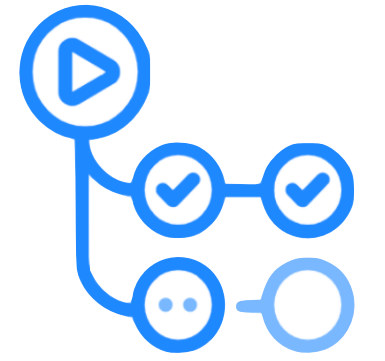
 ci.yml

 nightly-security-scan.yml

 release.yml

GitHub Actions: A Primer

Example Repository



GitHub Actions

"did I break something?"

runs on every commit / PR

```
on:  
  push:  
    branches: [main]  
  pull_request:  
    branches: [main]
```

▼  .github/workflows

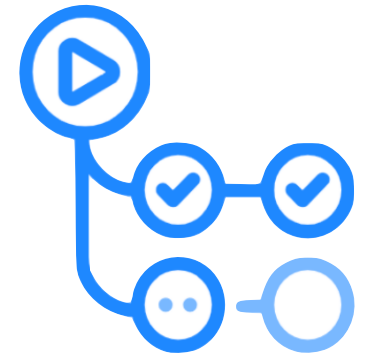
 ci.yml

 nightly-security-scan.yml

 release.yml

GitHub Actions: A Primer

Example Repository



GitHub Actions

also: external security threats!

useful for heavier security scans

runs on a schedule

▼ .github/workflows

ci.yml

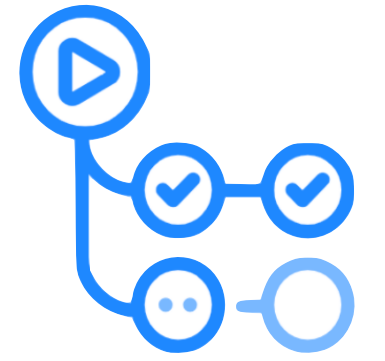
nightly-security-scan.yml

release.yml

```
on:
  schedule:
    # cron syntax: minute hour day month day-of-week
    # this runs every day at 02:00 UTC
    - cron: '0 2 * * *'
  # also allow manual runs from the Actions tab
  workflow_dispatch:
```

GitHub Actions: A Primer

Example Repository



GitHub Actions

also: external security threats!

useful for heavier security scans

runs on a schedule

▼  .github/workflows

 ci.yml

 nightly-security-scan.yml

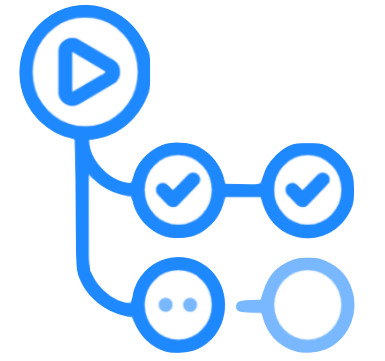
 release.yml

```
on:  
  schedule:  
    # cron syntax: minute hour day month day-of-week  
    # this runs every day at 02:00 UTC  
    - cron: '0 2 * * *'  
  # also allow manual runs from the Actions tab  
  workflow_dispatch:
```

can also be run manually




GitHub Actions: A Primer

Example Repository



GitHub Actions

▼  .github/workflows

-  ci.yml
-  nightly-security-scan.yml
-  release.yml

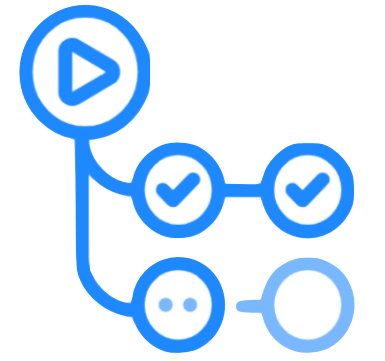
SBOM generation, signing, ...

runs when a new tag is pushed

```
on:  
  push:  
    tags:  
      - 'v*' # e.g. v1.0.0, v2.3.1
```

GitHub Actions: A Primer

Example Workflow



GitHub Actions

ci.yml

```
name: CI

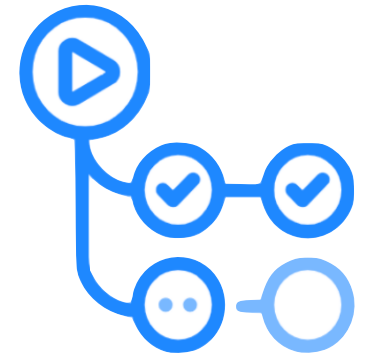
on:
  push:
    branches: [main]
  pull_request:
    branches: [main]

permissions:
  contents: read
  security-events: write
  actions: read

jobs: ...
```

GitHub Actions: A Primer

Example Workflow



GitHub Actions

ci.yml

name: CI

Workflow name for the GitHub Actions UI

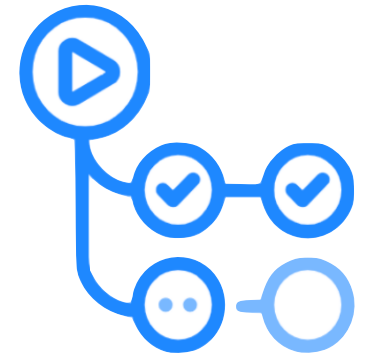
```
on:
  push:
    branches: [main]
  pull_request:
    branches: [main]

permissions:
  contents: read
  security-events: write
  actions: read

jobs: ...
```

GitHub Actions: A Primer

Example Workflow



GitHub Actions

ci.yml

```
name: CI

on:
  push:
    branches: [main]
  pull_request:
    branches: [main]

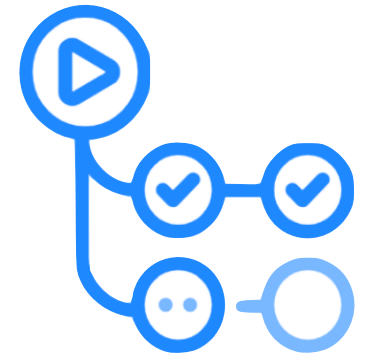
permissions:
  contents: read
  security-events: write
  actions: read

jobs: ...
```

Workflow trigger(s)

GitHub Actions: A Primer

Example Workflow



GitHub Actions

ci.yml

```
name: CI

on:
  push:
    branches: [main]
  pull_request:
    branches: [main]
```

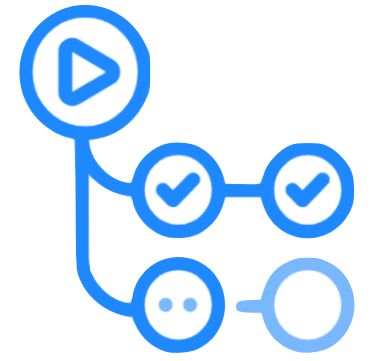
```
permissions:
  contents: read
  security-events: write
  actions: read
```

```
jobs: ...
```

Workflow permissions

GitHub Actions: A Primer

Example Workflow



GitHub Actions

ci.yml

```
name: CI

on:
  push:
    branches: [main]
  pull_request:
    branches: [main]

permissions:
  contents: read
  security-events: write
  actions: read
```

jobs: ...

Multiple jobs per workflow

GitHub Actions

Example Workflow

ci.yml

```
jobs:
  lint:
    name: Lint (cppcheck)
    runs-on: ubuntu-latest
    steps: ...

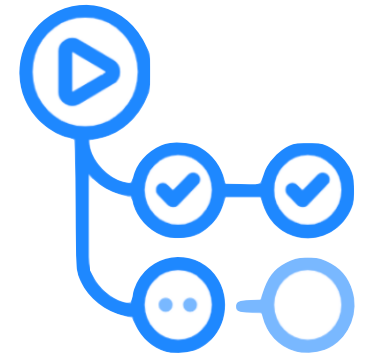
  secrets-scan:
    name: Secrets scan (gitleaks)
    runs-on: ubuntu-latest
    steps: ...

  build:
    name: Build firmware
    runs-on: ubuntu-latest
    steps: ...

  test:
    name: Unit tests
    needs: build
    runs-on: ubuntu-latest
    steps: ...

  security-report:
    name: Notify on security failure
    needs: [lint, test, secrets-scan]
    if: failure()
    runs-on: ubuntu-latest
    steps: ...
```

ner



GitHub Actions

GitHub Actions

Example Workflow

ci.yml

```
jobs:
  lint:
    name: Lint (cppcheck)
    runs-on: ubuntu-latest
    steps: ...

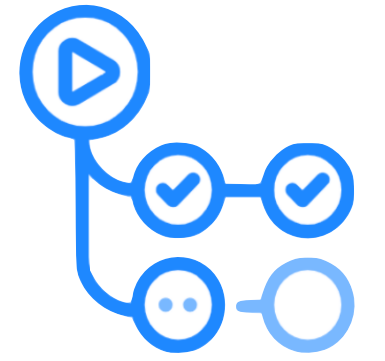
  secrets-scan:
    name: Secrets scan (gitleaks)
    runs-on: ubuntu-latest
    steps: ...

  build:
    name: Build firmware
    runs-on: ubuntu-latest
    steps: ...

  test:
    name: Unit tests
    needs: build
    runs-on: ubuntu-latest
    steps: ...

  security-report:
    name: Notify on security failure
    needs: [lint, test, secrets-scan]
    if: failure()
    runs-on: ubuntu-latest
    steps: ...
```

5 jobs



GitHub Actions

GitHub Actions

Example Workflow

ci.yml

```
jobs:
  lint:
    name: Lint (cppcheck)
    runs-on: ubuntu-latest
    steps: ...

  secrets-scan:
    name: Secrets scan (gitleaks)
    runs-on: ubuntu-latest
    steps: ...

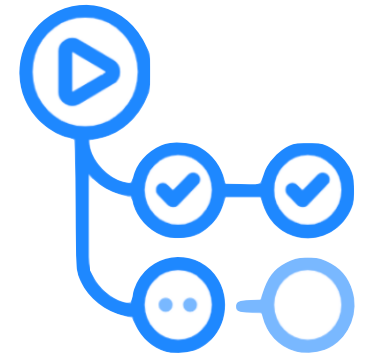
  build:
    name: Build firmware
    runs-on: ubuntu-latest
    steps: ...

  test:
    name: Unit tests
    needs: build
    runs-on: ubuntu-latest
    steps: ...

  security-report:
    name: Notify on security failure
    needs: [lint, test, secrets-scan]
    if: failure()
    runs-on: ubuntu-latest
    steps: ...
```

runner

each job has its own runner (provided by GitHub)



GitHub Actions

GitHub Actions

Example Workflow

ci.yml

```
jobs:
  lint:
    name: Lint (cppcheck)
    runs-on: ubuntu-latest
    steps: ...

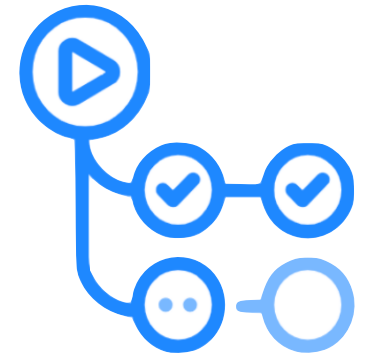
  secrets-scan:
    name: Secrets scan (gitleaks)
    runs-on: ubuntu-latest
    steps: ...

  build:
    name: Build firmware
    runs-on: ubuntu-latest
    steps: ...

  test:
    name: Unit tests
    needs: build
    runs-on: ubuntu-latest
    steps: ...

  security-report:
    name: Notify on security failure
    needs: [lint, test, secrets-scan]
    if: failure()
    runs-on: ubuntu-latest
    steps: ...
```

runner



GitHub Actions

each job has its own runner (provided by GitHub)

want to self-host instead?

<https://docs.github.com/en/actions/concepts/runners/self-hosted-runners>



GitHub Actions

Example Workflow

ci.yml

```
jobs:
  lint:
    name: Lint (cppcheck)
    runs-on: ubuntu-latest
    steps: ...

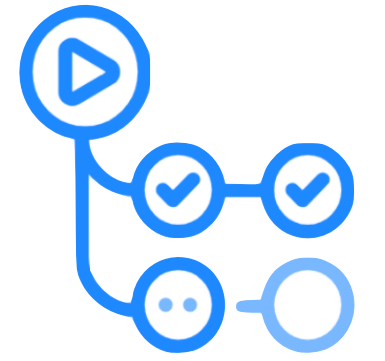
  secrets-scan:
    name: Secrets scan (gitleaks)
    runs-on: ubuntu-latest
    steps: ...

  build:
    name: Build firmware
    runs-on: ubuntu-latest
    steps: ...

  test:
    name: Unit tests
    needs: build
    runs-on: ubuntu-latest
    steps: ...

  security-report:
    name: Notify on security failure
    needs: [lint, test, secrets-scan]
    if: failure()
    runs-on: ubuntu-latest
    steps: ...
```

ner



GitHub Actions

By default, all jobs run in parallel ...

GitHub Actions

Example Workflow

ci.yml

```
jobs:
  lint:
    name: Lint (cppcheck)
    runs-on: ubuntu-latest
    steps: ...

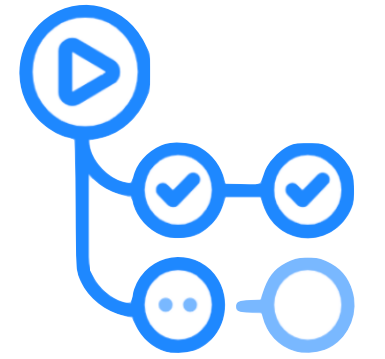
  secrets-scan:
    name: Secrets scan (gitleaks)
    runs-on: ubuntu-latest
    steps: ...

  build:
    name: Build firmware
    runs-on: ubuntu-latest
    steps: ...

  test:
    name: Unit tests
    needs: build
    runs-on: ubuntu-latest
    steps: ...

  security-report:
    name: Notify on security failure
    needs: [lint, test, secrets-scan]
    if: failure()
    runs-on: ubuntu-latest
    steps: ...
```

ner



GitHub Actions

By default, all jobs run in parallel ...

... unless job dependencies are explicitly specified

GitHub Actions

Example Workflow

ci.yml

```
jobs:
  lint:
    name: Lint (cppcheck)
    runs-on: ubuntu-latest
    steps: ...

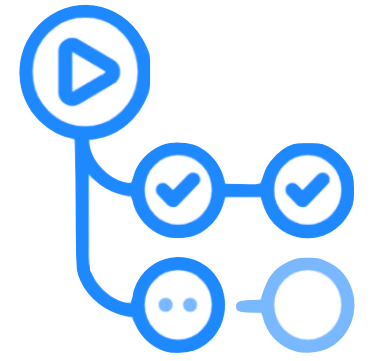
  secrets-scan:
    name: Secrets scan (gitleaks)
    runs-on: ubuntu-latest
    steps: ...

  build:
    name: Build firmware
    runs-on: ubuntu-latest
    steps: ...

  test:
    name: Unit tests
    needs: build
    runs-on: ubuntu-latest
    steps: ...

  security-report:
    name: Notify on security failure
    needs: [lint, test, secrets-scan]
    if: failure()
    runs-on: ubuntu-latest
    steps: ...
```

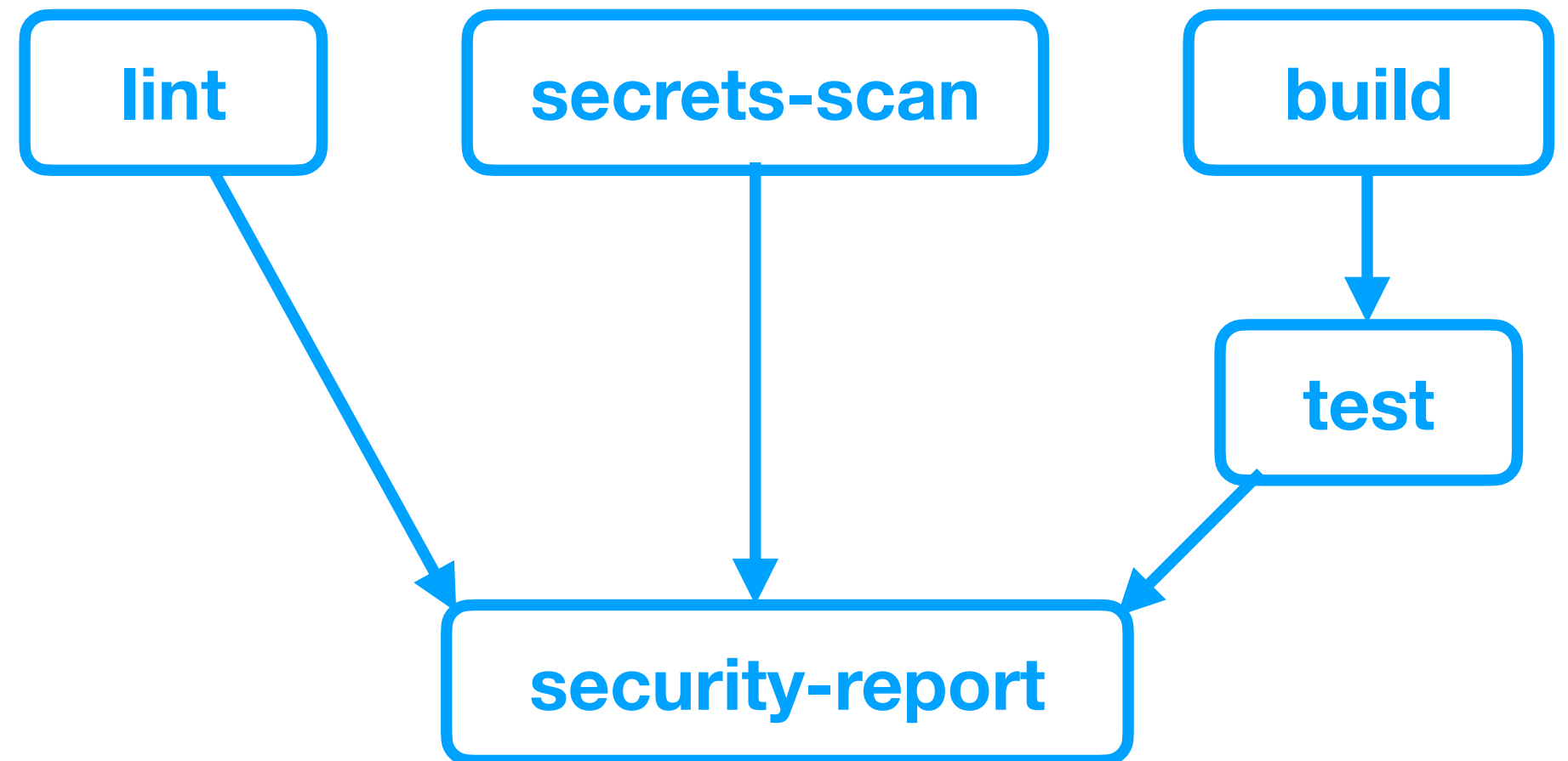
ner



GitHub Actions

By default, all jobs run in parallel ...

... unless job dependencies are explicitly specified



GitHub Actions

Example Workflow

ci.yml

```
jobs:
  lint:
    name: Lint (cppcheck)
    runs-on: ubuntu-latest
    steps: ...

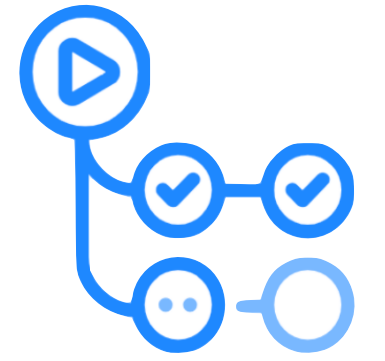
  secrets-scan:
    name: Secrets scan (gitleaks)
    runs-on: ubuntu-latest
    steps: ...

  build:
    name: Build firmware
    runs-on: ubuntu-latest
    steps: ...

  test:
    name: Unit tests
    needs: build
    runs-on: ubuntu-latest
    steps: ...

  security-report:
    name: Notify on security failure
    needs: [lint, test, secrets-scan]
    if: failure()
    runs-on: ubuntu-latest
    steps: ...
```

ner



GitHub Actions

conditional job execution

“Send a notification to Slack if one of the security checks fails”

GitHub Actions

Example Workflow

ci.yml

```
jobs:
  lint:
    name: Lint (cppcheck)
    runs-on: ubuntu-latest
    steps: ...

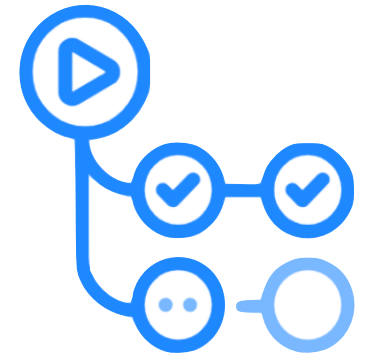
  secrets-scan:
    name: Secrets scan (gitleaks)
    runs-on: ubuntu-latest
    steps: ...

  build:
    name: Build firmware
    runs-on: ubuntu-latest
    steps: ...

  test:
    name: Unit tests
    needs: build
    runs-on: ubuntu-latest
    steps: ...

  security-report:
    name: Notify on security failure
    needs: [lint, test, secrets-scan]
    if: failure()
    runs-on: ubuntu-latest
    steps: ...
```

Multiple steps per job



GitHub Actions

GitHub Actions

Example Workflow

ci.yml

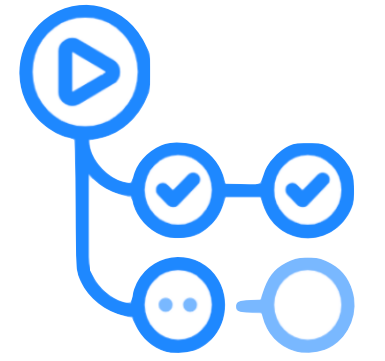
```
jobs:
  sast: ...
  secrets-scan: ...

  build:
    name: Build firmware
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - name: Set up Python
        uses: actions/setup-python@v5
        with:
          python-version: '3.11'
      - name: Install PlatformIO
        run: pip install --upgrade platformio
      - name: Build
        run: pio run
      - name: Upload firmware artifact
        uses: actions/upload-artifact@v4
        with:
          name: firmware
          path: .pio/build/**/firmware.bin

  test: ...

  security-report: ...
```

er



GitHub Actions

GitHub Actions

Example Workflow

ci.yml

```
jobs:
  sast: ...

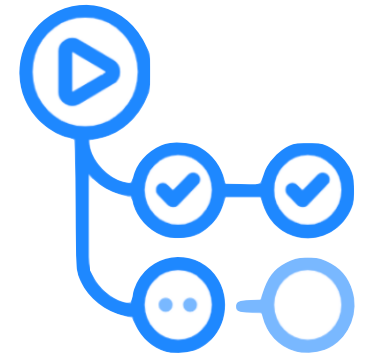
  secrets-scan: ...

  build:
    name: Build firmware
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - name: Set up Python
        uses: actions/setup-python@v5
        with:
          python-version: '3.11'
      - name: Install PlatformIO
        run: pip install --upgrade platformio
      - name: Build
        run: pio run
      - name: Upload firmware artifact
        uses: actions/upload-artifact@v4
        with:
          name: firmware
          path: .pio/build/**/firmware.bin

  test: ...

  security-report: ...
```

er



GitHub Actions

5 steps in the `build` job

steps run sequentially on the same runner

GitHub Actions

Example Workflow

ci.yml

```
jobs:
  sast: ...

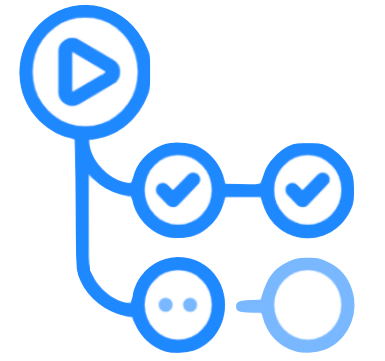
  secrets-scan: ...

  build:
    name: Build firmware
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - name: Set up Python
        uses: actions/setup-python@v5
        with:
          python-version: '3.11'
      - name: Install PlatformIO
        run: pip install --upgrade platformio
      - name: Build
        run: pio run
      - name: Upload firmware artifact
        uses: actions/upload-artifact@v4
        with:
          name: firmware
          path: .pio/build/**/firmware.bin

  test: ...

  security-report: ...
```

er



GitHub Actions

(optional) step name for the GitHub Actions UI

GitHub Actions

Example Workflow

ci.yml

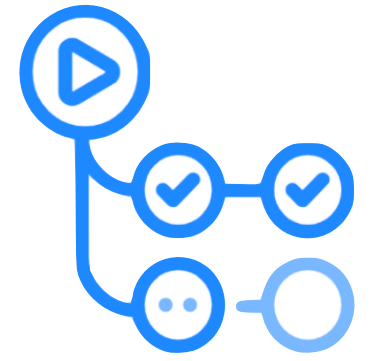
```
jobs:
  sast: ...
  secrets-scan: ...

  build:
    name: Build firmware
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - name: Set up Python
        uses: actions/setup-python@v5
        with:
          python-version: '3.11'
      - name: Install PlatformIO
        run: pip install --upgrade platformio
      - name: Build
        run: pio run
      - name: Upload firmware artifact
        uses: actions/upload-artifact@v4
        with:
          name: firmware
          path: .pio/build/**/firmware.bin

  test: ...

  security-report: ...
```

er



GitHub Actions

2 kinds of steps ...

GitHub Actions

Example Workflow

ci.yml

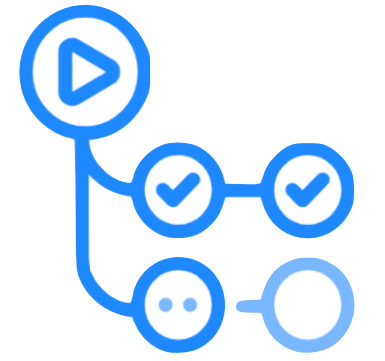
```
jobs:
  sast: ...
  secrets-scan: ...

  build:
    name: Build firmware
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - name: Set up Python
        uses: actions/setup-python@v5
        with:
          python-version: '3.11'
      - name: Install PlatformIO
        run: pip install --upgrade platformio
      - name: Build
        run: pio run
      - name: Upload firmware artifact
        uses: actions/upload-artifact@v4
        with:
          name: firmware
          path: .pio/build/**/firmware.bin

  test: ...

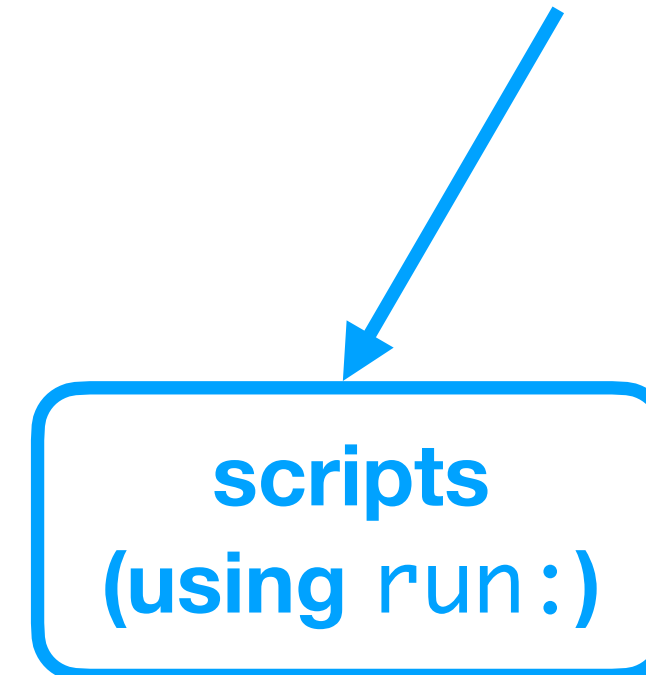
  security-report: ...
```

er



GitHub Actions

2 kinds of steps ...



GitHub Actions

Example Workflow

ci.yml

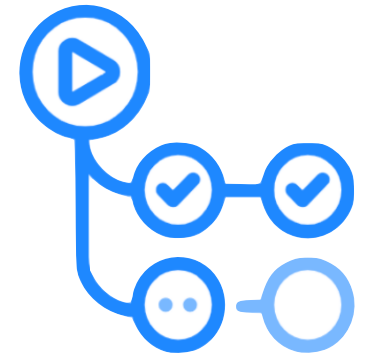
```
jobs:
  sast: ...
  secrets-scan: ...

  build:
    name: Build firmware
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - name: Set up Python
        uses: actions/setup-python@v5
        with:
          python-version: '3.11'
      - name: Install PlatformIO
        run: pip install --upgrade platformio
      - name: Build
        run: pio run
      - name: Upload firmware artifact
        uses: actions/upload-artifact@v4
        with:
          name: firmware
          path: .pio/build/**/firmware.bin

  test: ...

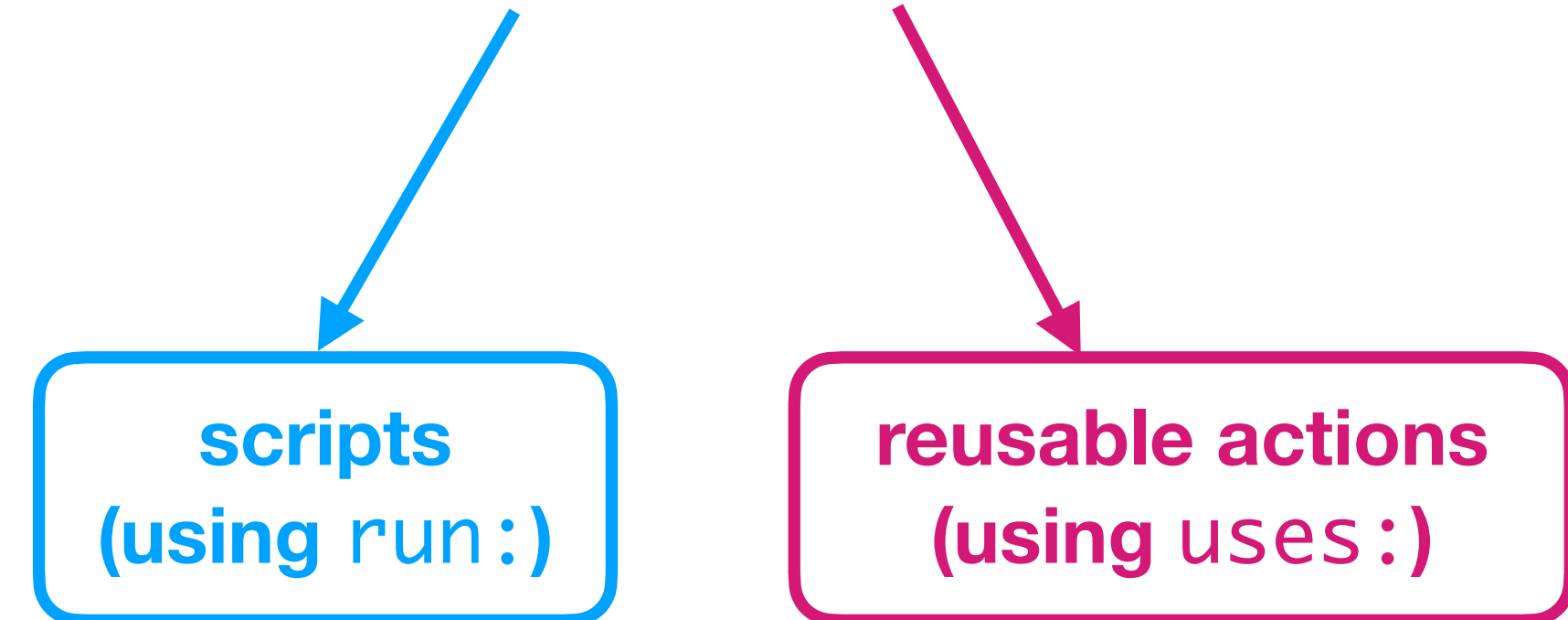
  security-report: ...
```

er



GitHub Actions

2 kinds of steps ...



GitHub Actions

Example Workflow

ci.yml

```
jobs:
  sast: ...
  secrets-scan: ...

  build:
    name: Build firmware
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - name: Set up Python
        uses: actions/setup-python@v5
        with:
          python-version: '3.11'
      - name: Install PlatformIO
        run: pip install --upgrade platformio
      - name: Build
        run: pio run
      - name: Upload firmware artifact
        uses: actions/upload-artifact@v4
        with:
          name: firmware
          path: .pio/build/**/firmware.bin

  test: ...

  security-report: ...
```

pin a version

uses: actions/setup-python@v5
with:
python-version: '3.11'

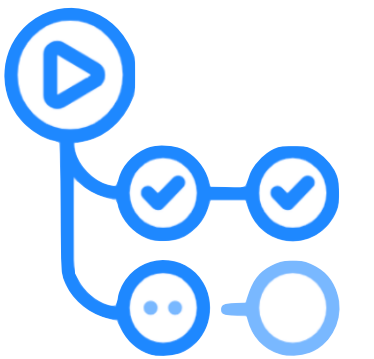
run: pip install --upgrade platformio

run: pio run

uses: actions/upload-artifact@v4
with:
name: firmware
path: .pio/build/**/firmware.bin

pass arguments

er



GitHub Actions

2 kinds of steps ...

scripts
(using run:)

reusable actions
(using uses:)

GitHub Actions

Example Workflow

ci.yml

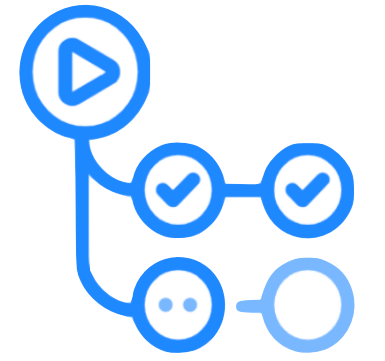
```
jobs:
  sast: ...
  secrets-scan: ...

  build:
    name: Build firmware
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - name: Set up Python
        uses: actions/setup-python@v5
        with:
          python-version: '3.11'
      - name: Install PlatformIO
        run: pip install --upgrade platformio
      - name: Build
        run: pio run
      - name: Upload firmware artifact
        uses: actions/upload-artifact@v4
        with:
          name: firmware
          path: .pio/build/**/firmware.bin

  test: ...

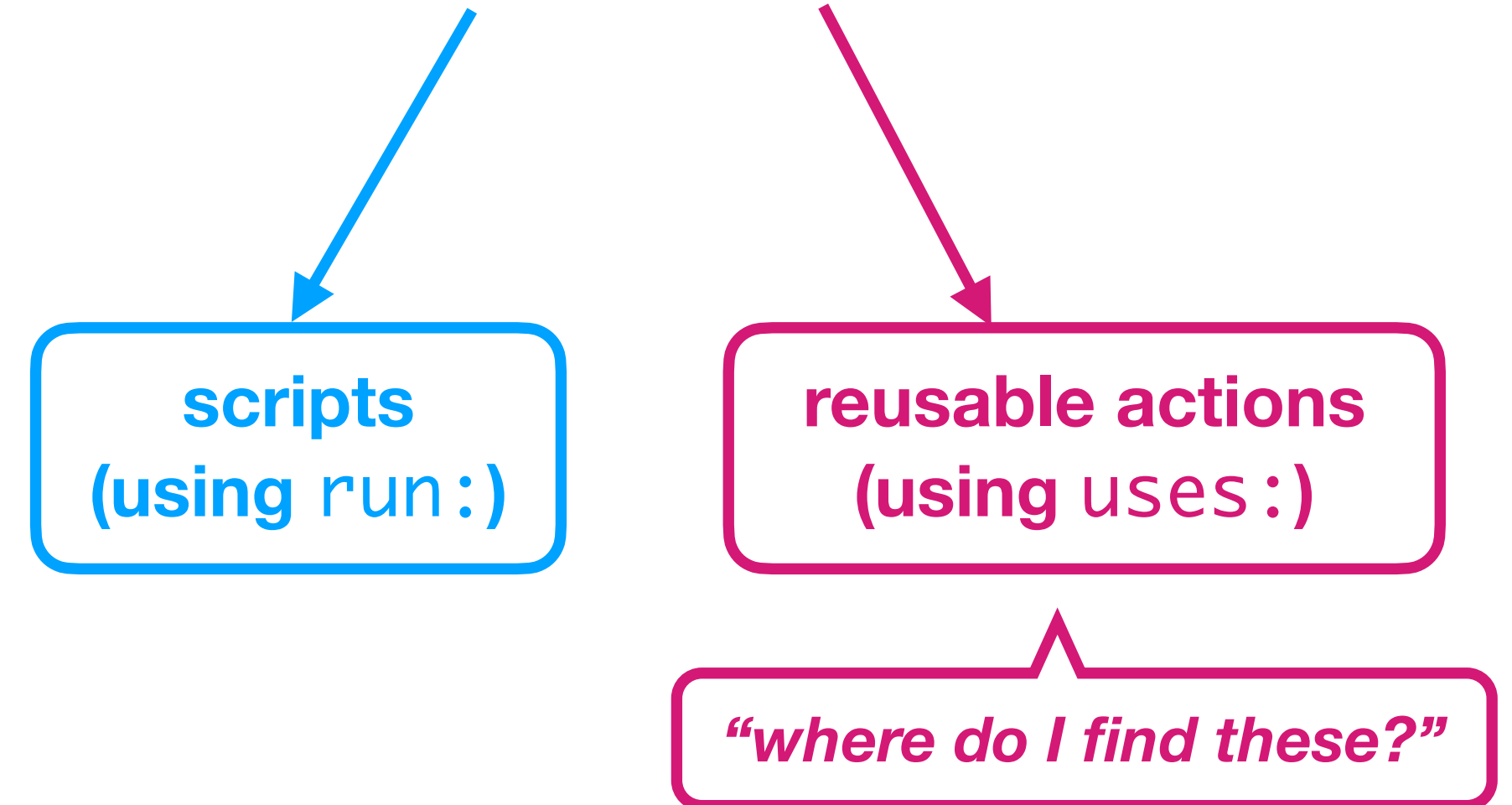
  security-report: ...
```

er



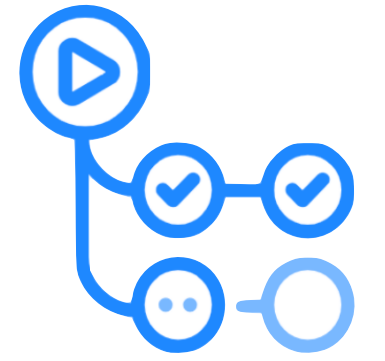
GitHub Actions

2 kinds of steps ...

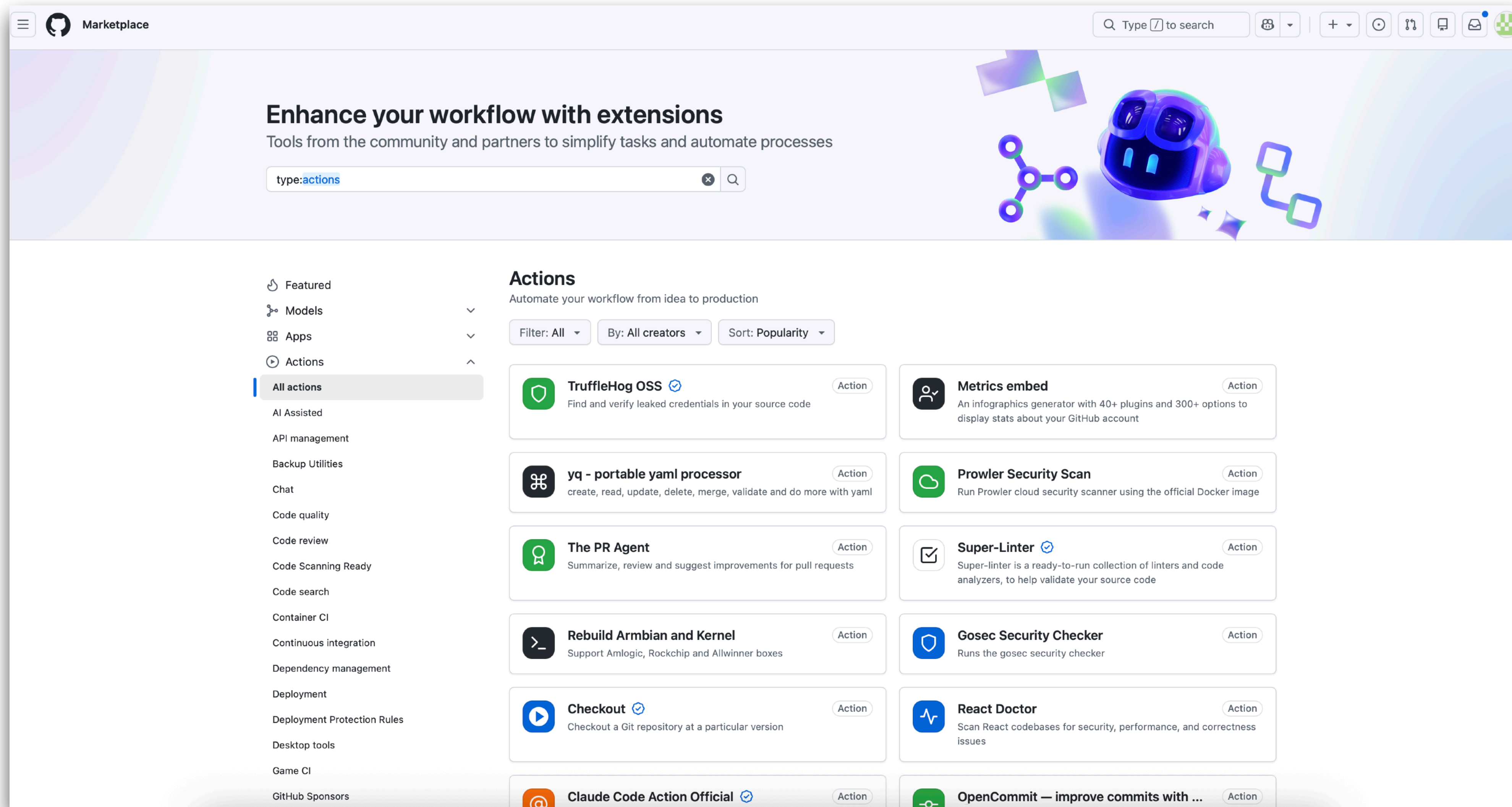


GitHub Actions: A Primer

GitHub Marketplace



GitHub Actions



The screenshot shows the GitHub Marketplace interface for Actions. At the top, there's a search bar with the text "type:actions" and a search icon. Below the search bar, a banner reads "Enhance your workflow with extensions" with the subtitle "Tools from the community and partners to simplify tasks and automate processes".

On the left side, there's a navigation menu with categories like "Featured", "Models", "Apps", "Actions", and "All actions". The "All actions" category is currently selected.

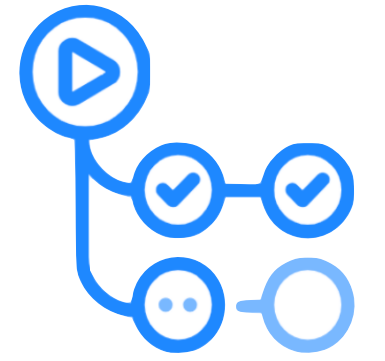
The main content area is titled "Actions" and includes a sub-header "Automate your workflow from idea to production". Below this, there are filter and sort options: "Filter: All", "By: All creators", and "Sort: Popularity".

The main content area displays a grid of action cards, each with an icon, a title, a description, and an "Action" button. The visible actions include:

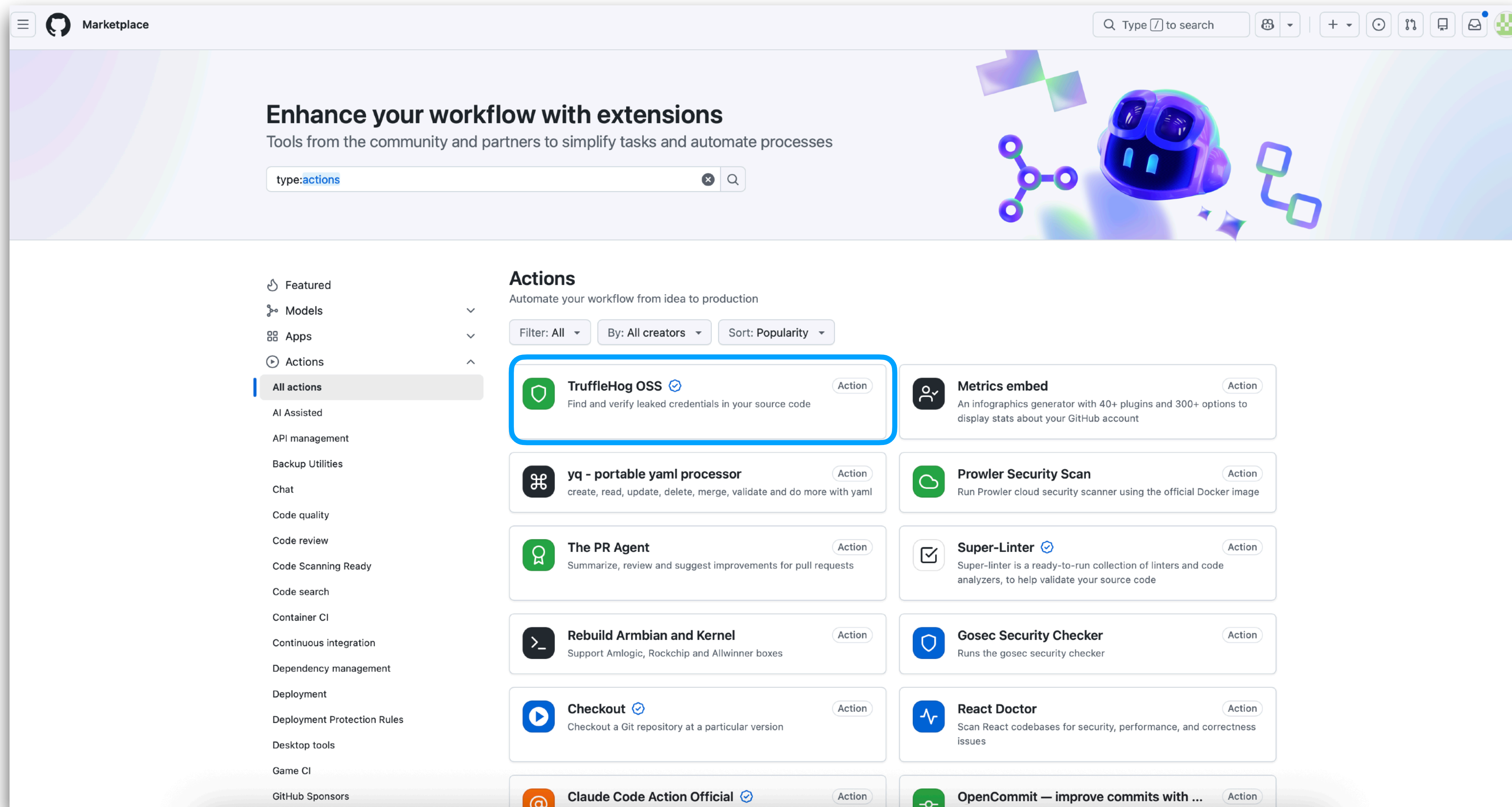
- TruffleHog OSS**: Find and verify leaked credentials in your source code.
- Metrics embed**: An infographics generator with 40+ plugins and 300+ options to display stats about your GitHub account.
- yq - portable yaml processor**: create, read, update, delete, merge, validate and do more with yaml.
- Prowler Security Scan**: Run Prowler cloud security scanner using the official Docker image.
- The PR Agent**: Summarize, review and suggest improvements for pull requests.
- Super-Linter**: Super-linter is a ready-to-run collection of linters and code analyzers, to help validate your source code.
- Rebuild Armbian and Kernel**: Support Amlogic, Rockchip and Allwinner boxes.
- Gosec Security Checker**: Runs the gosec security checker.
- Checkout**: Checkout a Git repository at a particular version.
- React Doctor**: Scan React codebases for security, performance, and correctness issues.
- Claude Code Action Official**: (partially visible)
- OpenCommit — improve commits with ...**: (partially visible)

GitHub Actions: A Primer

GitHub Marketplace



GitHub Actions



The screenshot shows the GitHub Marketplace interface for Actions. At the top, there's a search bar with the text "type:actions" and a search icon. Below the search bar, the main heading reads "Enhance your workflow with extensions" with the subtitle "Tools from the community and partners to simplify tasks and automate processes".

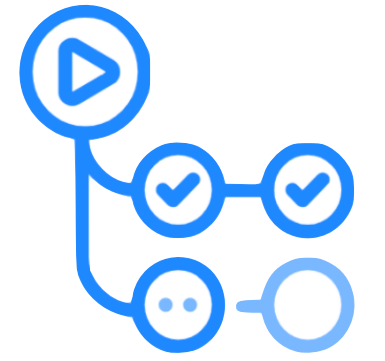
On the left side, there's a navigation menu with categories like "Featured", "Models", "Apps", "Actions", and "All actions". The "All actions" category is currently selected.

The main content area is titled "Actions" and includes a sub-header "Automate your workflow from idea to production". Below this, there are filter and sort options: "Filter: All", "By: All creators", and "Sort: Popularity".

The main content area displays a grid of action cards. The first card, "TruffleHog OSS", is highlighted with a blue border. It features a green shield icon and the description "Find and verify leaked credentials in your source code". Other visible cards include "Metrics embed", "Prowler Security Scan", "The PR Agent", "Super-Linter", "Rebuild Armbian and Kernel", "Gosec Security Checker", "Checkout", "React Doctor", and "Claude Code Action Official".

GitHub Actions: A Primer

GitHub Marketplace



GitHub Actions

The screenshot displays the GitHub Marketplace interface. On the left, a sidebar lists various categories under 'Actions', with 'All actions' selected. The main content area shows the 'TruffleHog OSS' action page. At the top, it features the TruffleHog logo (a pig and a key) and the text 'TruffleHog' and 'Find leaked credentials.'. Below this, there are statistics: 'go report A+', 'license AGPL-3.0', and 'Total Detectors 870'. A 'Now Scanning' section lists supported integrations: GitHub, GitLab, Git, S3, GCS, CircleCI, Docker, Travis CI, Postman, Jenkins, and Elasticsearch. The right sidebar contains details about the action, including 'About', 'Verified' status, 'Tags' (continuous-integration, security), 'Contributors' (188), and 'Resources' (256 issues, 188 pull requests).

Marketplace

Enhance your workflow with
Tools from the community and partners to

type:actions

Featured

Models

Apps

Actions

All actions

AI Assisted

API management

Backup Utilities

Chat

Code quality

Code review

Code Scanning Ready

Code search

Container CI

Continuous integration

Dependency management

Deployment

Deployment Protection Rules

Desktop tools

Game CI

GitHub Sponsors

Automate

Filter: A

TruffleHog OSS Actions

Star 26.1K Use latest version

About

Find and verify leaked credentials in your source code

v3.95.2 Latest

By truffelsecurity

Verified

GitHub has manually verified the creator of the action as an official partner organization. For more info see [About badges in GitHub Marketplace](#).

Tags 2

continuous-integration security

Contributors 188

+ 174 contributors

Resources

Open an issue 256

Pull requests 188

View source code

Security policy

Report abuse

TruffleHog

Find leaked credentials.

go report A+

license AGPL-3.0

Total Detectors 870

Now Scanning

GitHub GitLab Git S3 GCS CircleCI Docker Travis CI Postman Jenkins Elasticsearch

...and more

To learn more about TruffleHog and its features and capabilities, visit our [product page](#).

TruffleHog Enterprise

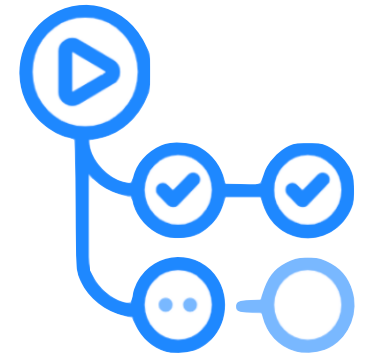
Are you interested in continuously monitoring Git, Jira, Slack, Confluence, Microsoft Teams, Sharepoint (and more) for credentials? We have an enterprise product that can help! Learn more at <https://trufflesecurity.com/trufflehog-enterprise>.

We take the revenue from the enterprise product to fund more awesome open source projects that the whole community can benefit from.

TruffleHog OSS is not certified by GitHub. It is provided by a third-party and is governed by separate terms of service, privacy policy, and support documentation.

GitHub Actions: A Primer

GitHub Marketplace



GitHub Actions

The screenshot displays the GitHub Marketplace interface. On the left, the 'Marketplace' header is visible. The main content area shows a search for 'type:actions' and a list of categories including 'Featured', 'Models', 'Apps', and 'Actions'. The 'Actions' category is selected, and the 'TruffleHog OSS' action is highlighted. The right sidebar shows the 'TruffleHog OSS' action details, including a star count of 26.1K, a 'Use latest version' button, and a description: 'Find and verify leaked credentials in your source code'. The 'About' section mentions that GitHub has manually verified the creator as an official partner organization. The 'Verified' section states that GitHub has manually verified the creator. The 'Tags' section includes 'continuous-integration' and 'security'. The 'Contributors' section shows 188 contributors. The 'Resources' section includes links to 'Open an issue', 'Pull requests', 'View source code', 'Security policy', and 'Report abuse'. The 'Now Scanning' section lists various CI/CD tools supported by TruffleHog, including GitHub, GitLab, Git, S3, GCS, CircleCI, Docker, Travis CI, Postman, Jenkins, and Elasticsearch. The 'TruffleHog Enterprise' section is also visible at the bottom.

Marketplace

Enhance your workflow with
Tools from the community and partners to

type:actions

Featured
Models
Apps
Actions
All actions
AI Assisted
API management
Backup Utilities
Chat
Code quality
Code review
Code Scanning Ready
Code search
Container CI
Continuous integration
Dependency management
Deployment
Deployment Protection Rules
Desktop tools
Game CI
GitHub Sponsors

Action

Automate

Filter: A

TruffleHog OSS Actions

Star 26.1K Use latest version

About

Find and verify leaked credentials in your source code

v3.95.2 Latest

By truffelsecurity

Verified

GitHub has manually verified the creator of the action as an official partner organization. For more info see [About badges in GitHub Marketplace](#).

Tags 2

continuous-integration security

Contributors 188

+ 174 contributors

Resources

Open an issue 256

Pull requests 188

View source code

Security policy

Report abuse

TruffleHog

Find leaked credentials.

go report A+
license AGPL-3.0
Total Detectors 870

Now Scanning

GitHub GitLab Git S3 GCS CircleCI Docker Travis CI Postman Jenkins Elasticsearch

...and more

To learn more about TruffleHog and its features and capabilities, visit our [product page](#).

TruffleHog Enterprise

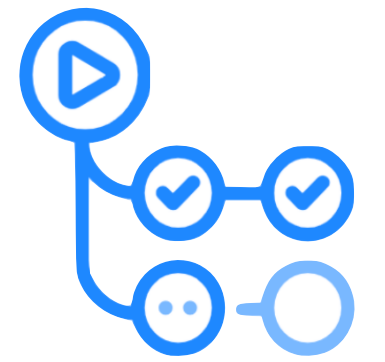
Are you interested in continuously monitoring Git, Jira, Slack, Confluence, Microsoft Teams, Sharepoint (and more) for credentials? We have an enterprise product that can help! Learn more at <https://trufflesecurity.com/trufflehog-enterprise>.

We take the revenue from the enterprise product to fund more awesome open source projects that the whole community can benefit from.

TruffleHog OSS is not certified by GitHub. It is provided by a third-party and is governed by separate terms of service, privacy policy, and support documentation.

GitHub Actions: A Primer

GitHub Marketplace



GitHub Actions

A composite screenshot of the GitHub Marketplace interface. On the left, the 'Marketplace' header is visible with a search bar containing 'type:actions'. A sidebar on the left lists various categories, with 'All actions' highlighted. The main content area shows the 'TruffleHog OSS' action card. The card includes the TruffleHog logo, the name 'TruffleHog OSS', a description 'Find and verify leaked credentials in your source code', a star count of 26.1K, and a 'Use latest version' button. A modal window is overlaid on the card, displaying the installation instructions: 'Copy and paste the following snippet into your .yaml file.' followed by a code block:

```
- name: TruffleHog OSS
  uses: trufflesecurity/trufflehog@v3.95.2
```

 Below the code block is a link to 'Learn more about this action in trufflesecurity/trufflehog'. The background shows the 'Now Search' section with various CI/CD tool icons and a 'TruffleHog Enterprise' section.

GitHub Actions

Example Workflow

ci.yml

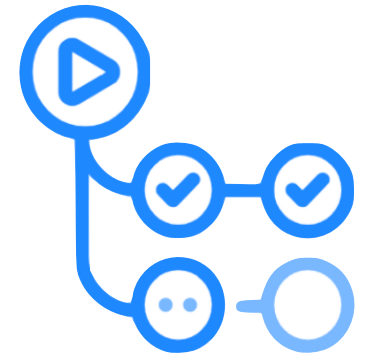
```
jobs:
  sast: ...
  secrets-scan: ...

  build:
    name: Build firmware
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - name: Set up Python
        uses: actions/setup-python@v5
        with:
          python-version: '3.11'
      - name: Install PlatformIO
        run: pip install --upgrade platformio
      - name: Build
        run: pio run
      - name: Upload firmware artifact
        uses: actions/upload-artifact@v4
        with:
          name: firmware
          path: .pio/build/**/firmware.bin

  test: ...

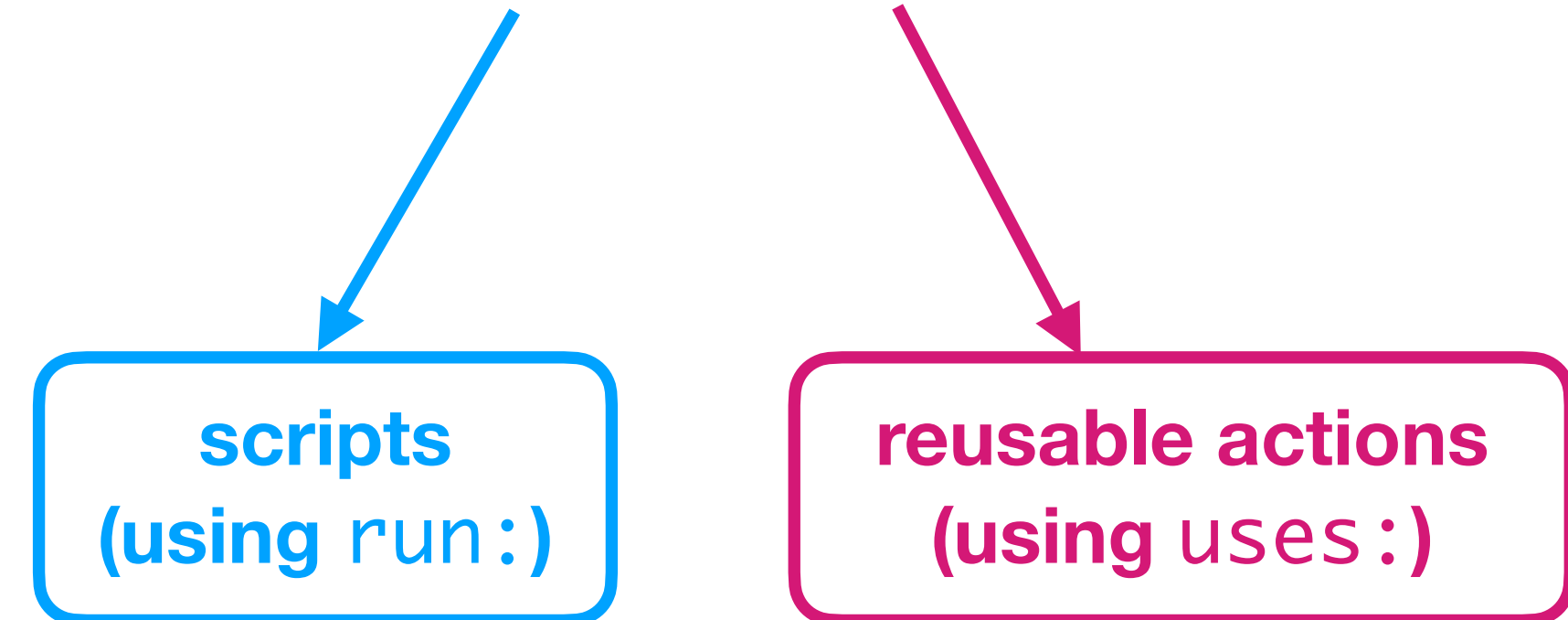
  security-report: ...
```

er



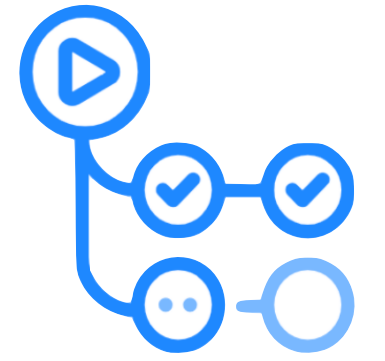
GitHub Actions

2 kinds of steps ...



GitHub Actions

Example Workflow



GitHub Actions

ci.yml

```
jobs:
  sast: ...
  secrets-scan: ...

  build:
    name: Build firmware
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - name: Set up Python
        uses: actions/setup-python@v5
        with:
          python-version: '3.11'
      - name: Install PlatformIO
        run: pip install --upgrade platformio
      - name: Build
        run: pio run
      - name: Upload firmware artifact
        uses: actions/upload-artifact@v4
        with:
          name: firmware
          path: .pio/build/**/firmware.bin

  test: ...

  security-report: ...
```

er

checkout code repository

setup Python 3.11

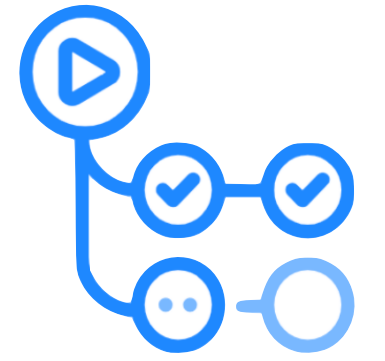
install platformIO

build firmware using platformIO

upload artifact

GitHub Actions

Example Workflow



GitHub Actions

ci.yml

```
jobs:
  sast: ...
  secrets-scan: ...

  build:
    name: Build firmware
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
      - name: Set up Python
        uses: actions/setup-python@v5
        with:
          python-version: '3.11'
      - name: Install PlatformIO
        run: pip install --upgrade platformio
      - name: Build
        run: pio run
      - name: Upload firmware artifact
        uses: actions/upload-artifact@v4
        with:
          name: firmware
          path: .pio/build/**/firmware.bin

  test: ...
  security-report: ...
```

er

checkout code repository

setup Python 3.11

install platformIO

build firmware using platformIO

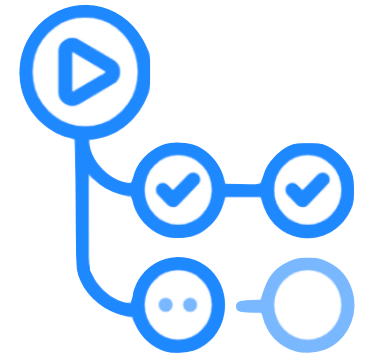
upload artifact

... used for **sharing files** between jobs/workflows

... used as part of the **results** of a workflow

GitHub Actions: A Primer

Workflow Results



GitHub Actions

Navigation: <> Code | Pull requests | Agents | **Actions** | Projects | Wiki | Security and quality 15 | Insights | Settings

Actions | New workflow

All workflows

CI (demo)

CI (demo) ci.yml

Filter workflow runs

14 workflow runs

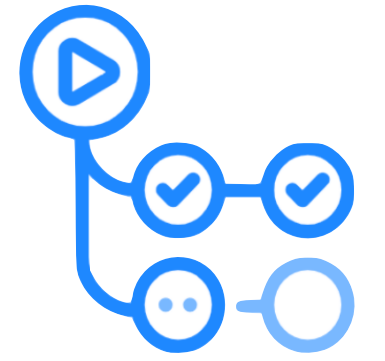
	Event	Status	Branch	Actor
✓ [BUGFIX] Fix off-by-one error CI (demo) #14: Commit 3afe072 pushed by noahvanes	main	1 minute ago 21s		...
✗ Introduce a bug that makes tests fail CI (demo) #13: Commit a4d34ec pushed by noahvanes	main	8 minutes ago 32s		...
✓ Update CI workflow CI (demo) #12: Commit 34d532f pushed by noahvanes	main	10 minutes ago 14s		...
✓ Rename CI workflow CI (demo) #11: Commit 6708da6 pushed by noahvanes	main	12 minutes ago 12s		...
✓ Add example for CI workflow CI (demo) #10: Commit 71eebb1 pushed by noahvanes	main	14 minutes ago 12s		...
✓ Add example for CI workflow CI (demo) #9: Commit c64be82 pushed by noahvanes	main	16 minutes ago 2m 51s		...
✓ Rename workflow CI (demo) #8: Commit d253503 pushed by noahvanes	main	17 minutes ago 2m 48s		...
✓ Refactor CI workflow and add placeholder steps		52 minutes ago		...

Management

- Caches
- Attestations ↗
- Runners
- Usage metrics ↗
- Performance metrics ↗

GitHub Actions: A Primer

Workflow Results



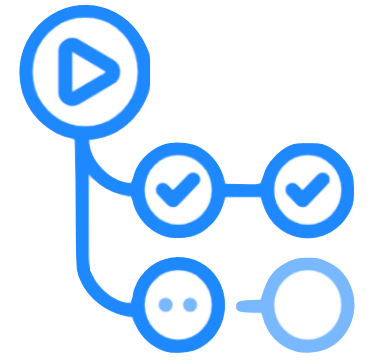
GitHub Actions

The screenshot shows the GitHub Actions interface for a workflow named "CI (demo)". The "Actions" tab is selected in the top navigation bar. In the left sidebar, the "CI (demo)" workflow is highlighted, with a callout box labeled "selected workflow". The main area displays a list of 14 workflow runs. A callout box labeled "workflow runs" points to the list. The runs are sorted by time, with the most recent at the top. The first run, "[BUGFIX] Fix off-by-one error", is successful. The second run, "Introduce a bug that makes tests fail", is failed. The remaining runs are successful. The table below summarizes the visible runs.

Event	Status	Branch	Actor	Time	Duration
[BUGFIX] Fix off-by-one error	Success	main	noahvanes	1 minute ago	21s
Introduce a bug that makes tests fail	Failure	main	noahvanes	8 minutes ago	32s
Update CI workflow	Success	main	noahvanes	10 minutes ago	14s
Rename CI workflow	Success	main	noahvanes	12 minutes ago	12s
Add example for CI workflow	Success	main	noahvanes	14 minutes ago	12s
Add example for CI workflow	Success	main	noahvanes	16 minutes ago	2m 51s
Rename workflow	Success	main	noahvanes	17 minutes ago	2m 48s
Refactor CI workflow and add placeholder steps	Success	main	noahvanes	52 minutes ago	

GitHub Actions: A Primer

Workflow Results



GitHub Actions

Navigation: <> Code | Pull requests | Agents | **Actions** | Projects | Wiki | Security and quality 15 | Insights | Settings

Actions New workflow

All workflows

CI (demo)

CI (demo) [ci.yml](#) ...

14 workflow runs

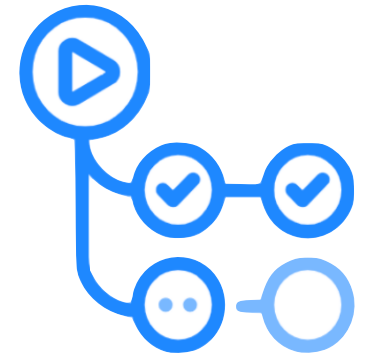
	Event	Status	Branch	Actor
✓ [BUGFIX] Fix off-by-one error CI (demo) #14: Commit 3afe072 pushed by noahvanes		Success	main	
✗ Introduce a bug that makes tests fail CI (demo) #13: Commit a4d34ec pushed by noahvanes		Failure	main	
✓ Update CI workflow CI (demo) #12: Commit 34d532f pushed by noahvanes		Success	main	
✓ Rename CI workflow CI (demo) #11: Commit 6708da6 pushed by noahvanes		Success	main	
✓ Add example for CI workflow CI (demo) #10: Commit 71eebb1 pushed by noahvanes		Success	main	
✓ Add example for CI workflow CI (demo) #9: Commit c64be82 pushed by noahvanes		Success	main	
✓ Rename workflow CI (demo) #8: Commit d253503 pushed by noahvanes		Success	main	
✓ Refactor CI workflow and add placeholder steps CI (demo) #7: Commit ... pushed by noahvanes		Success	main	

Management

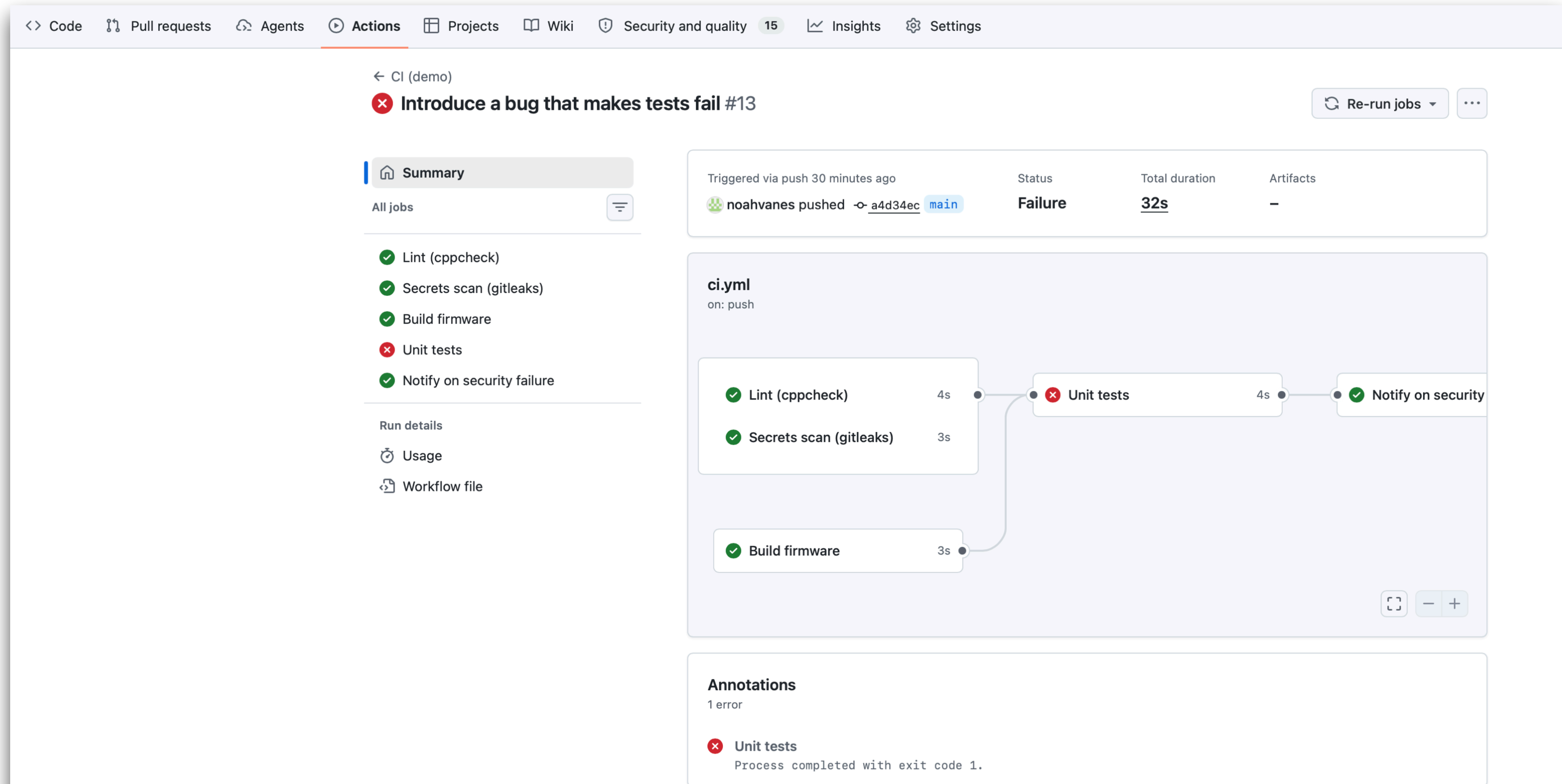
- Caches
- Attestations ↗
- Runners
- Usage metrics ↗
- Performance metrics ↗

GitHub Actions: A Primer

Workflow Results



GitHub Actions



The screenshot shows the GitHub Actions interface for a workflow named "CI (demo)". The workflow run is titled "Introduce a bug that makes tests fail #13" and is in a "Failure" state. The run was triggered by a push from user "noahvanes" 30 minutes ago. The total duration of the run is 32 seconds. The workflow file is "ci.yml" and it runs on the "push" event.

The workflow steps are as follows:

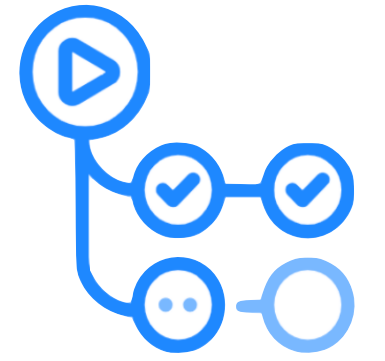
- Lint (cppcheck) - 4s - Success
- Secrets scan (gitleaks) - 3s - Success
- Build firmware - 3s - Success
- Unit tests - 4s - Failure
- Notify on security failure - Success

The "Unit tests" step failed with the error: "Process completed with exit code 1." The "Notify on security failure" step is also shown as successful.

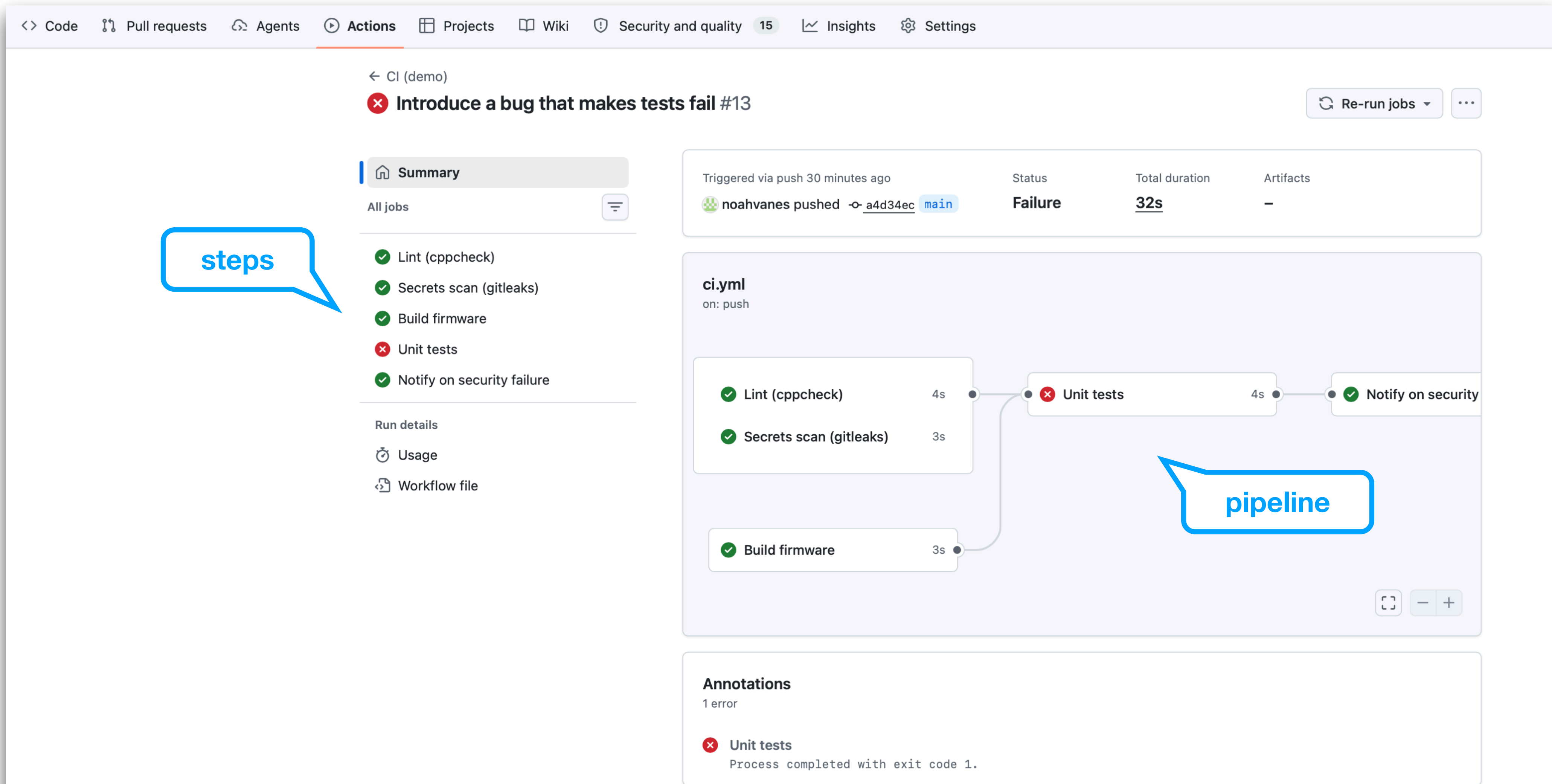
The interface includes a navigation bar with links for Code, Pull requests, Agents, Actions, Projects, Wiki, Security and quality, Insights, and Settings. The left sidebar shows a summary of the workflow jobs, with "Unit tests" highlighted as failed. The right sidebar shows the workflow graph and annotations.

GitHub Actions: A Primer

Workflow Results



GitHub Actions



The screenshot shows the GitHub Actions interface for a workflow named "CI (demo)". The workflow run is titled "Introduce a bug that makes tests fail #13" and has a status of "Failure". It was triggered via a push 30 minutes ago by user "noahvanes" on the "main" branch. The total duration of the run is 32 seconds. The workflow file is "ci.yml" and it runs on "push".

The workflow steps are listed in the "Summary" section and visualized in the "pipeline" diagram:

- Lint (cppcheck) - 4s (Success)
- Secrets scan (gitleaks) - 3s (Success)
- Build firmware - 3s (Success)
- Unit tests - 4s (Failure)
- Notify on security failure (Success)

The "Annotations" section shows one error: "Unit tests" failed with the message "Process completed with exit code 1.".

Navigation links at the top include: Code, Pull requests, Agents, Actions, Projects, Wiki, Security and quality (15), Insights, and Settings.

Additional navigation links on the left include: Summary, All jobs, Run details, Usage, and Workflow file.

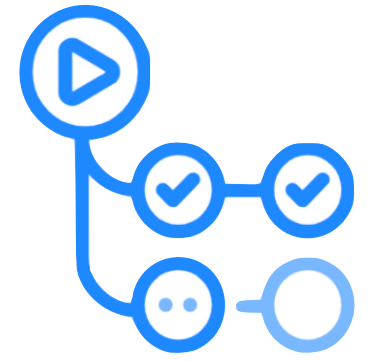
Buttons for "Re-run jobs" and a menu icon are located in the top right of the workflow run summary.

steps

pipeline

GitHub Actions: A Primer

Workflow Results



GitHub Actions

Navigation: <> Code | Pull requests | Agents | **Actions** | Projects | Wiki | Security and quality 15 | Insights | Settings

Actions New workflow

All workflows

CI (demo)

CI (demo) [ci.yml](#) ...

14 workflow runs

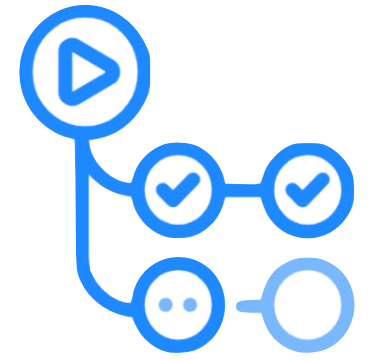
	Event	Status	Branch	Actor
✓ [BUGFIX] Fix off-by-one error CI (demo) #14: Commit 3afe072 pushed by noahvanes		Success	main	
✗ Introduce a bug that makes tests fail CI (demo) #13: Commit a4d34ec pushed by noahvanes		Failure	main	
✓ Update CI workflow CI (demo) #12: Commit 34d532f pushed by noahvanes		Success	main	
✓ Rename CI workflow CI (demo) #11: Commit 6708da6 pushed by noahvanes		Success	main	
✓ Add example for CI workflow CI (demo) #10: Commit 71eebb1 pushed by noahvanes		Success	main	
✓ Add example for CI workflow CI (demo) #9: Commit c64be82 pushed by noahvanes		Success	main	
✓ Rename workflow CI (demo) #8: Commit d253503 pushed by noahvanes		Success	main	
✓ Refactor CI workflow and add placeholder steps CI (demo) #7: Commit ... pushed by noahvanes		Success	main	

Management

- Caches
- Attestations ↗
- Runners
- Usage metrics ↗
- Performance metrics ↗

GitHub Actions: A Primer

Workflow Results



GitHub Actions

← CI (demo)

🟢 [BUGFIX] Fix off-by-one error #14 Re-run all jobs ⋮

Summary

All jobs ☰

- 🟢 Lint (cppcheck)
- 🟢 Secrets scan (gitleaks)
- 🟢 Build firmware
- 🟢 Unit tests
- 🔴 Notify on security failure

Run details

- 🕒 Usage
- 📄 Workflow file

Triggered via push 29 minutes ago

Triggered via	Status	Total duration	Artifacts
🌱 noahvanes pushed -o- 3afe072 main	Success	21s	-

ci.yml
on: push

```
graph LR; A[Lint (cppcheck) 3s] --> B[Unit tests 4s]; C[Secrets scan (gitleaks) 2s] --> B; D[Build firmware 3s] --> B; B --> E[Notify on security failure 0s]
```

🟢 Lint (cppcheck) 3s

🟢 Secrets scan (gitleaks) 2s

🟢 Build firmware 3s

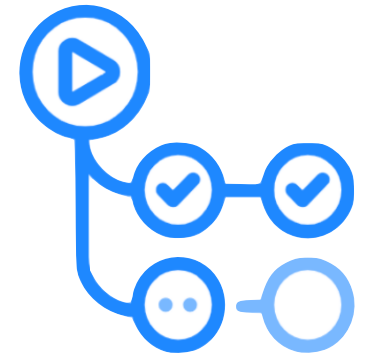
🟢 Unit tests 4s

🔴 Notify on security failure 0s

🔍 - +

GitHub Actions: A Primer

Workflow Results



GitHub Actions

← CI (demo)

🟢 [BUGFIX] Fix off-by-one error #14 Re-run all jobs ⋮

Summary

All jobs

- 🟢 Lint (cppcheck)
- 🟢 Secrets scan (gitleaks)
- 🟢 Build firmware
- 🟢 Unit tests
- 🔴 Notify on security failure

Run details

- 🕒 Usage
- 📄 Workflow file

Triggered via push 29 minutes ago

noahvanes pushed -o- 3afe072 main

Status	Total duration	Artifacts
Success	21s	-

ci.yml
on: push

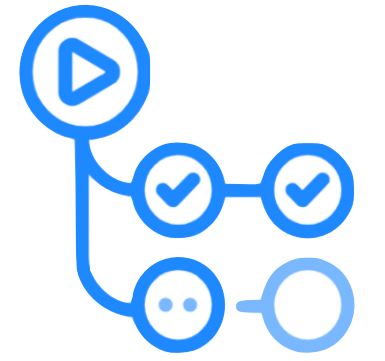
```
graph LR; Lint[Lint (cppcheck) 3s] --> Unit[Unit tests 4s]; Secrets[Secrets scan (gitleaks) 2s] --> Unit; Build[Build firmware 3s] --> Unit; Unit --> Notify[Notify on security failure 0s];
```

skipped

🔍 - +

GitHub Actions: A Primer

Workflow Results



GitHub Actions

← CI (demo)

✓ [BUGFIX] Fix off-by-one error #14 Re-run all jobs ⋮

Summary

All jobs

- ✓ Lint (cppcheck)
- ✓ Secrets scan (gitleaks)
- ✓ Build firmware
- ✓ Unit tests
- ⊗ Notify on security failure

Run details

- Usage
- Workflow file

Triggered via push 29 minutes ago

Status: **Success** Total duration: **21s** Artifacts: -

noahvanes pushed -> `3afe072` `main`

ci.yml
on: push

skipped

Lint (cppcheck) 3s

Secrets scan (gitleaks) 2s

Unit tests 4s

Notify on security failure 0s

Artifacts

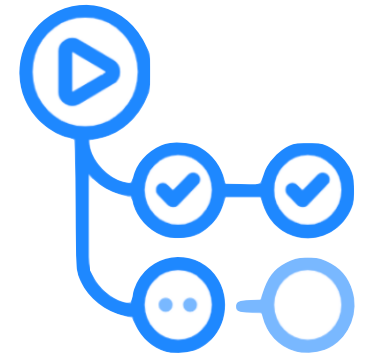
Produced during runtime

artifacts produced by the workflow

Name	Size	Digest		
cppcheck.sarif	1.11 KB	sha256:a40658a53b3ead959530ff0db86c802d79ee791e5b0e11a3dee5a57...	📄	🗑️
firmware-m5stick	703 KB	sha256:14415b067c2edf88301f04f0af8267e6714e2b8ce18501d6eedf73f...	📄	🗑️
gitleaks-results.sarif	6.6 KB	sha256:15542a6c983bf7675113b5b9107aa59cbc226f38ddb10cdd1b08724...	📄	🗑️
sca-reports	4.41 KB	sha256:3beec8ca4e20e84f0b5c570755e2d8c41ceff372f8c3ee413837113...	📄	🗑️

GitHub Actions: A Primer

Secrets Management

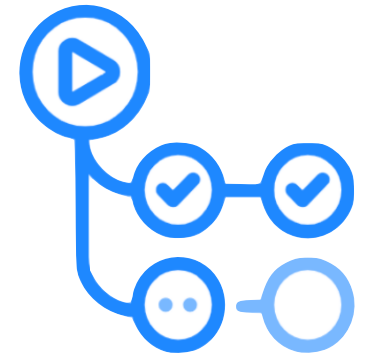


GitHub Actions

```
build:
  name: Build Firmware
  runs-on: ubuntu-latest
  steps:
    - name: Checkout code
      uses: actions/checkout@v3
    - name: Set up Python
      uses: actions/setup-python@v4
    - name: Install PlatformIO
      run: pip install platformio
    - name: Build firmware
      run: pio run --environment m5stick-c
      env:
        D_WIFI_SSID: "MyWifiName"
        D_WIFI_PASS: "MyWifiPassword"
        D_WIFI_USER: "MyUserName"
        GITHUB_TOKEN: "ghs_123456789"
    - name: Upload firmware binary
      uses: actions/upload-artifact@v4
      with:
        name: firmware-m5stick
        path: |
          .pio/build/m5stick-c/firmware.bin
          .pio/build/m5stick-c/partitions.bin
```

GitHub Actions: A Primer

Secrets Management



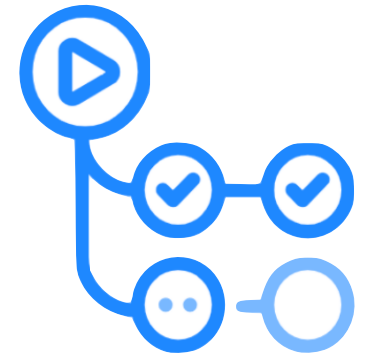
GitHub Actions

exposes secrets!

```
build:
  name: Build Firmware
  runs-on: ubuntu-latest
  steps:
    - name: Checkout code
      uses: actions/checkout@v3
    - name: Set up Python
      uses: actions/setup-python@v4
    - name: Install PlatformIO
      run: pip install platformio
    - name: Build firmware
      run: pio run --environment m5stick-c
      env:
        D_WIFI_SSID: "MyWifiName"
        D_WIFI_PASS: "MyWifiPassword"
        D_WIFI_USER: "MyUserName"
        GITHUB_TOKEN: "ghs_123456789"
    - name: Upload firmware binary
      uses: actions/upload-artifact@v4
      with:
        name: firmware-m5stick
        path: |
          .pio/build/m5stick-c/firmware.bin
          .pio/build/m5stick-c/partitions.bin
```

GitHub Actions: A Primer

Secrets Management



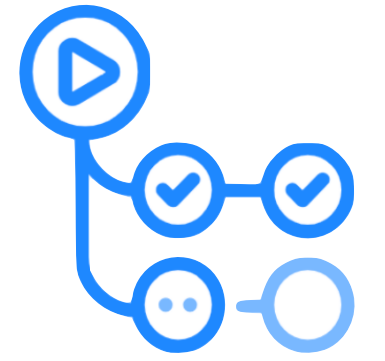
GitHub Actions

secrets are kept secret

```
build:
  name: Build Firmware
  runs-on: ubuntu-latest
  steps:
    - name: Checkout code
      uses: actions/checkout@v3
    - name: Set up Python
      uses: actions/setup-python@v4
    - name: Install PlatformIO
      run: pip install platformio
    - name: Build firmware
      run: pio run --environment m5stick-c
      env:
        D_WIFI_SSID: ${ secrets.WIFI_SSID }
        D_WIFI_PASS: ${ secrets.WIFI_PASS }
        D_WIFI_USER: ${ secrets.WIFI_USER }
        GITHUB_TOKEN: ${ secrets.GITHUB_TOKEN }
    - name: Upload firmware binary
      uses: actions/upload-artifact@v4
      with:
        name: firmware-m5stick
        path: |
          .pio/build/m5stick-c/firmware.bin
          .pio/build/m5stick-c/partitions.bin
```

GitHub Actions: A Primer

Secrets Management



GitHub Actions

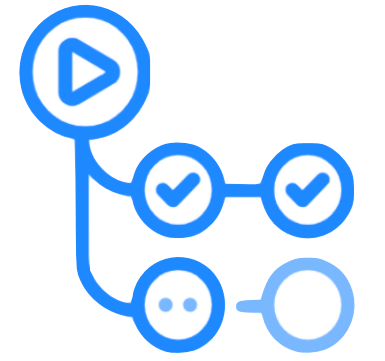
```
build:
  name: Build Firmware
  runs-on: ubuntu-latest
  steps:
    - name: Checkout code
      uses: actions/checkout@v3
    - name: Set up Python
      uses: actions/setup-python@v4
    - name: Install PlatformIO
      run: pip install platformio
    - name: Build firmware
      run: pio run --environment m5stick-c
      env:
        D_WIFI_SSID: ${ secrets.WIFI_SSID }
        D_WIFI_PASS: ${ secrets.WIFI_PASS }
        D_WIFI_USER: ${ secrets.WIFI_USER }
        GITHUB_TOKEN: ${ secrets.GITHUB_TOKEN }
    - name: Upload firmware binary
      uses: actions/upload-artifact@v4
      with:
        name: firmware-m5stick
        path: |
          .pio/build/m5stick-c/firmware.bin
          .pio/build/m5stick-c/partitions.bin
```

secrets are kept secret

implicit **secrets** object available

GitHub Actions: A Primer

Secrets Management



GitHub Actions

```
build:
  name: Build Firmware
  runs-on: ubuntu-latest
  steps:
    - name: Checkout code
      uses: actions/checkout@v3
    - name: Set up Python
      uses: actions/setup-python@v4
    - name: Install PlatformIO
      run: pip install platformio
    - name: Build firmware
      run: pio run --environment m5stick-c
  env:
    D_WIFI_SSID: ${ secrets.WIFI_SSID }
    D_WIFI_PASS: ${ secrets.WIFI_PASS }
    D_WIFI_USER: ${ secrets.WIFI_USER }
    GITHUB_TOKEN: ${ secrets.GITHUB_TOKEN }
    - name: Upload firmware binary
      uses: actions/upload-artifact@v4
      with:
        name: firmware-m5stick
        path: |
          .pio/build/m5stick-c/firmware.bin
          .pio/build/m5stick-c/partitions.bin
```

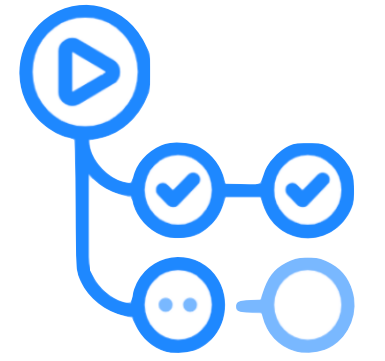
secrets are kept secret

implicit **secrets** object available

Where does this come from?

GitHub Actions: A Primer

Secrets Management



GitHub Actions

The screenshot shows the GitHub Actions settings page. A blue box labeled '1' highlights the 'Settings' tab in the top navigation bar. A blue box labeled '2' highlights the 'Secrets and variables' menu item under the 'Security' section. A blue box labeled '3' highlights the 'New repository secret' button. The page content includes a description of secrets and variables, a warning about collaborator access, and a table of repository secrets.

1

2

3

Code Issues Pull requests 6 Agents Actions Projects Wiki Security and quality Insights Settings

Security

- * Secrets and variables
- Actions

Actions secrets and variables

Secrets and variables allow you to manage reusable configuration data. Secrets are **encrypted** and are used for sensitive data. [Learn more about encrypted secrets](#). Variables are shown as plain text and are used for **non-sensitive** data. [Learn more about variables](#).

Anyone with collaborator access to this repository can use these secrets and variables for actions. They are not passed to workflows that are triggered by a pull request from a fork.

Secrets Variables

Repository secrets

Name	Last updated
WIFI_SSID	last month
WIFI_USER	last month

New repository secret

Ok, let's build a CI/CD pipeline!

A Simple Starting Point ...

Hands-on!

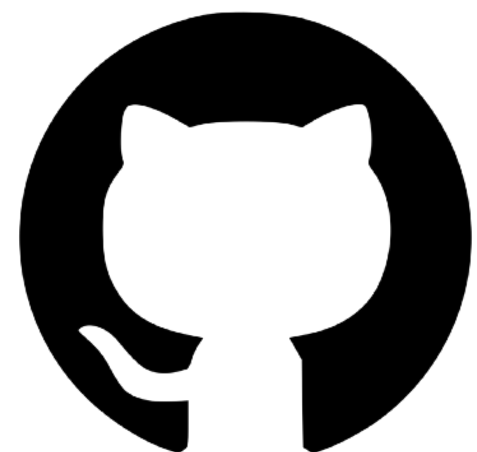
PlatformIO

<https://platformio.org>



M5StickC PLUS2

ESP32 device



<https://github.com/noahvanes/bugatti-workshop-demo-ex1>



Initial CI/CD Pipeline

.workflows/ci.yml

```
name: CI

on:
  push:
    branches: [main]
  pull_request:
    branches: [main]

permissions:
  contents: read
  security-events: write
  actions: read

jobs:

  build:
    name: Build Firmware
    runs-on: ubuntu-latest
    steps:
      - name: Checkout code
        uses: actions/checkout@v3
      - name: Set up Python
        uses: actions/setup-python@v4
      - name: Install PlatformIO
        run: pip install platformio
      - name: Build firmware
        run: pio run --environment m5stick-c
```

only contains a **build** job

Recall: What's in a CI/CD Pipeline?

Software Composition Analysis (SCA)

Dependency Scanning

SBOM Generation

Static Application Security Testing (SAST)

Dynamic Application Security Testing (DAST) / Fuzzing

Unit & Integration Testing

IaC Security Scanning

Build

Firmware Analysis

Compliance Checking

Secure Deployment

Secret Scanning

Artifact/Firmware Signing

A Security-Focused CI/CD Pipeline

Overview



A Security-Focused CI/CD Pipeline

Overview



Secret Scanning

detect accidental leakage of sensitive information (credentials, API keys, ...)

based on **pattern matching, ML, entropy analysis, ...**

Secret Scanning

detect accidental leakage of sensitive information (credentials, API keys, ...)

based on **pattern matching, ML, entropy analysis, ...**

➔ example

src/main.cpp

```
#define BITCOIN_PRICE_ENDPOINT "https://api.coingecko.com/api/v3/simple/price?ids=bitcoin&vs_currencies=usd"  
#define COINGECKO_API_KEY "CG-7kPq2Nm9Xc4vT8jY1wL6bZ3h"
```

```
HTTPClient http;  
http.setTimeout(5000);  
http.addHeader("x-cg-demo-api-key", COINGECKO_API_KEY);  
  
if (!http.begin(client, BITCOIN_PRICE_ENDPOINT)) {  
    Serial.println("Bitcoin fetch: Failed to begin HTTP");  
    return;  
}  
  
int httpCode = http.GET();
```

API key leaked in source code!

often happens in configuration files as well

Secret Scanning

detect accidental leakage of sensitive information (credentials, API keys, ...)

based on **pattern matching, ML, entropy analysis, ...**

➔ detection tools



Secret Scanning

detect accidental leakage of sensitive information (credentials, API keys, ...)

based on **pattern matching, ML, entropy analysis, ...**

➔ detection tools



free and **open-source**



enterprise

Secret Scanning

detect accidental leakage of sensitive information (credentials, API keys, ...)

based on **pattern matching, ML, entropy analysis, ...**

➔ detection tools



free and **open-source**

enterprise

GitHub Actions also has some (basic) **built-in secret scanning**

Secret Scanning

detect accidental leakage of sensitive information (credentials, API keys, ...)

based on **pattern matching, ML, entropy analysis, ...**

➔ detection tools



free and **open-source**

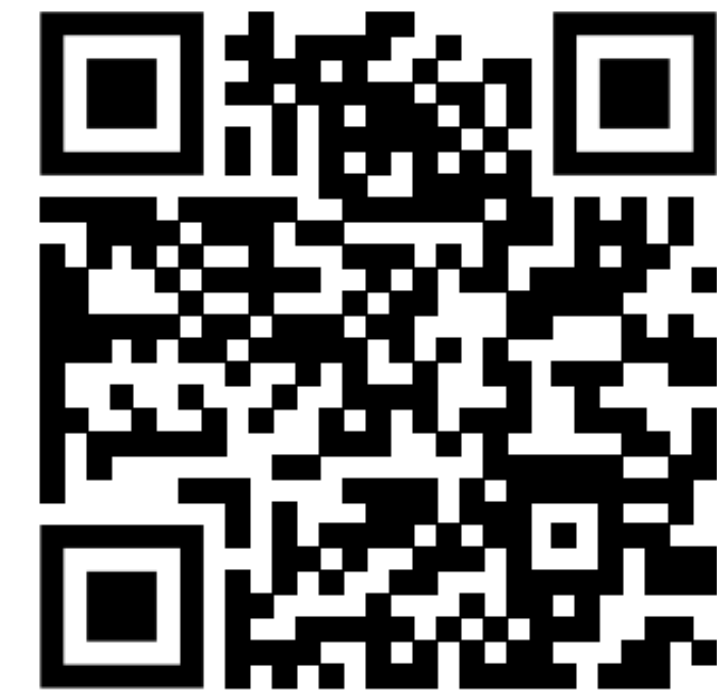
Adding Secret Scanning to the Pipeline

Hands-on Exercise



```
scan:  
  name: gitleaks  
  runs-on: ubuntu-latest  
  steps:  
    - uses: actions/checkout@v4  
      with:  
        fetch-depth: 0  
    - uses: gitleaks/gitleaks-action@v2  
  env:  
    GITHUB_TOKEN: ${ secrets.GITHUB_TOKEN }  
    GITLEAKS_LICENSE: ${ secrets.GITLEAKS_LICENSE } # Only required for Organizations
```

GitHub Actions Marketplace link



Adding Secret Scanning to the Pipeline

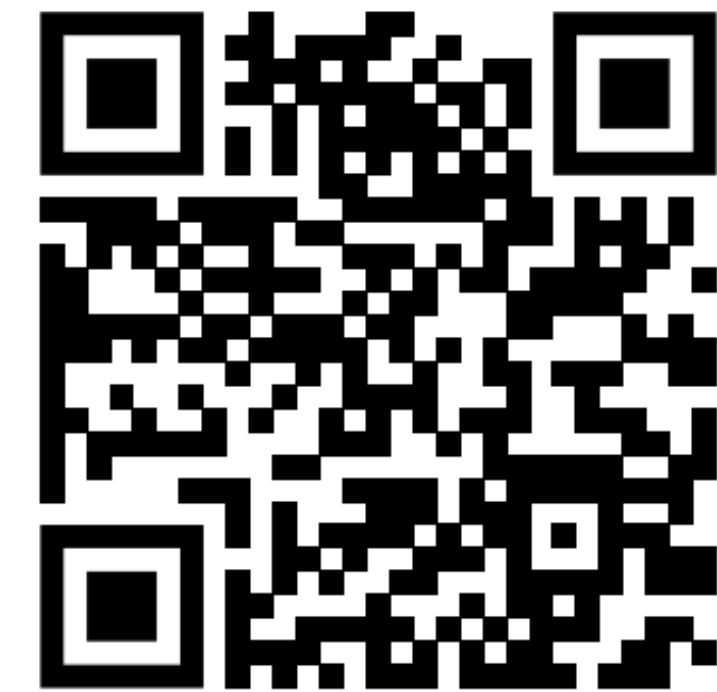
Hands-on Exercise



```
scan:  
  name: gitleaks  
  runs-on: ubuntu-latest  
  steps:  
    - uses: actions/checkout@v4  
      with:  
        fetch-depth: 0  
    - uses: gitleaks/gitleaks-action@v2  
  env:  
    GITHUB_TOKEN: ${ secrets.GITHUB_TOKEN }  
    GITLEAKS_LICENSE: ${ secrets.GITLEAKS_LICENSE } # Only required for Organizations
```

checkout all history (for all branches, tags, ...)

GitHub Actions Marketplace link



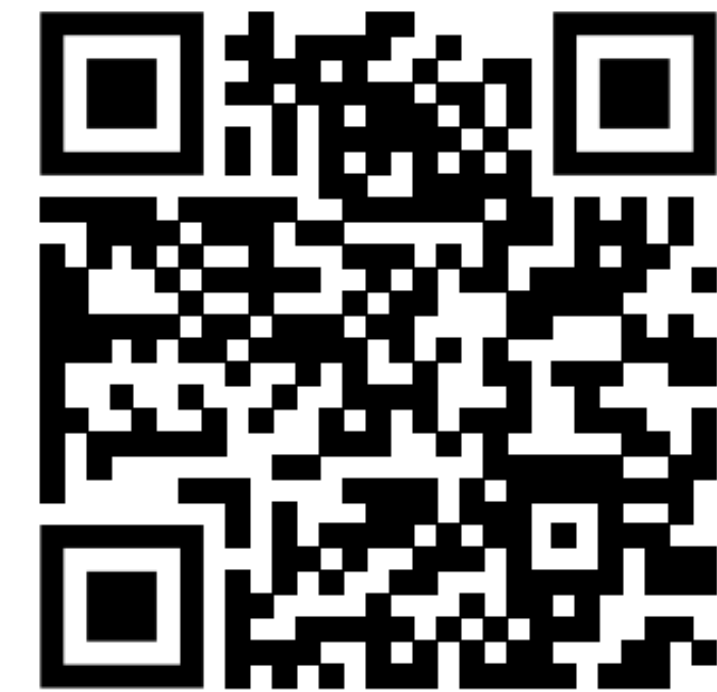
Adding Secret Scanning to the Pipeline

Hands-on Exercise



```
jobs:
  scan:
    name: gitleaks
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
        with:
          fetch-depth: 0
      - uses: gitleaks/gitleaks-action@v2
    env:
      GITHUB_TOKEN: ${ secrets.GITHUB_TOKEN }
      GITLEAKS_LICENSE: ${ secrets.GITLEAKS_LICENSE } # Only required for Organizations
```

GitHub Actions Marketplace link



Exercise: add a GitHub Actions job to the CI to detect exposed secrets in the comments branch

Adding Secret Scanning to the Pipeline

Discussion

Question: What to do when exposed secrets are detected?

Adding Secret Scanning to the Pipeline

Discussion

Question: Where to put the secret scanning job w.r.t. the build job?

Adding Secret Scanning to the Pipeline

Discussion

Question: How to avoid exposing secrets?

A Security-Focused CI/CD Pipeline

Overview



laC Security Scanning

infrastructure-as-code



ANSIBLE



HashiCorp

Terraform

laC Security Scanning

infrastructure-as-code

detect security misconfigurations, compliance violations, ...



ANSIBLE



HashiCorp

Terraform

laC Security Scanning

infrastructure-as-code

detect security misconfigurations, compliance violations, ...



ANSIBLE

SCAnsible

GASEL



HashiCorp

Terraform

tf-check

Terrascan

Snyk

IaC Security Scanning

infrastructure-as-code

detect security misconfigurations, compliance violations, ...

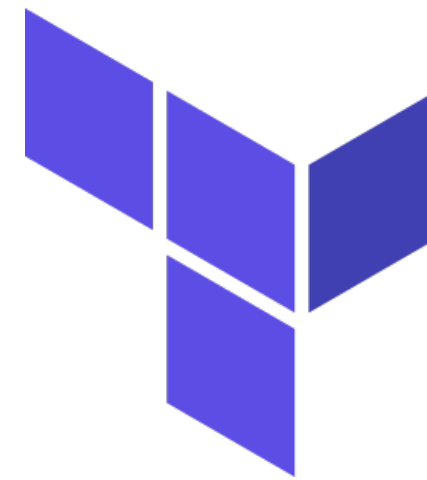


ANSIBLE

SCAnsible

GASEL

Interested? Let us know!



HashiCorp

Terraform

tf-check

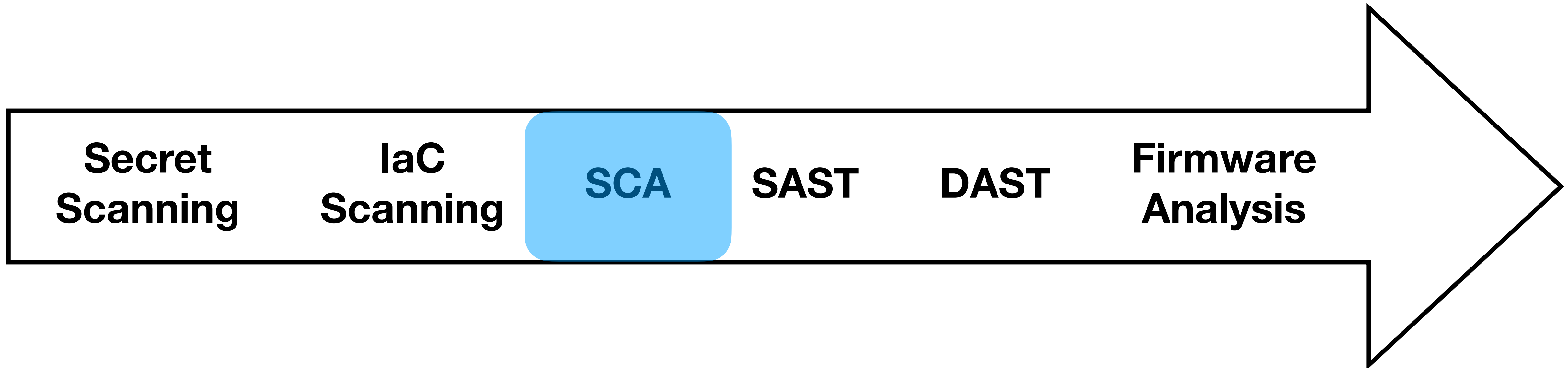
Terrascan

Snyk

Out-of-the-box actions available on the marketplace!

A Security-Focused CI/CD Pipeline

Overview



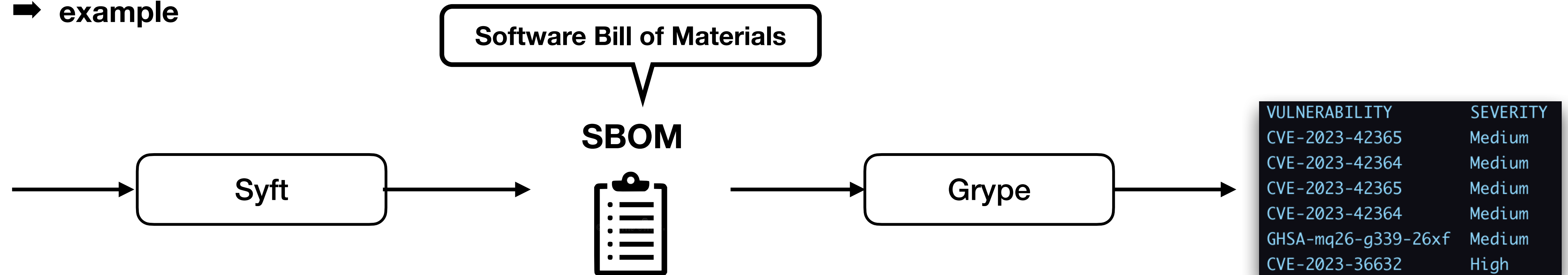
Software Composition Analysis (SCA)

scan dependencies for known vulnerabilities (CVEs)

Software Composition Analysis (SCA)

scan dependencies for known vulnerabilities (CVEs)

➔ example



Software Composition Analysis (SCA)

scan dependencies for known vulnerabilities (CVEs)

➔ example

Software Bill of Materials

SBOM

Syft

Grype

VULNERABILITY	SEVERITY
CVE-2023-42365	Medium
CVE-2023-42364	Medium
CVE-2023-42365	Medium
CVE-2023-42364	Medium
GHSA-mq26-g339-26xf	Medium
CVE-2023-36632	High

```
- uses: anchore/sbom-action@v0
with:
  format: cyclonedx-json
  output-file: sbom.json

- uses: anchore/scan-action@v3
with:
  sbom: sbom.json
```

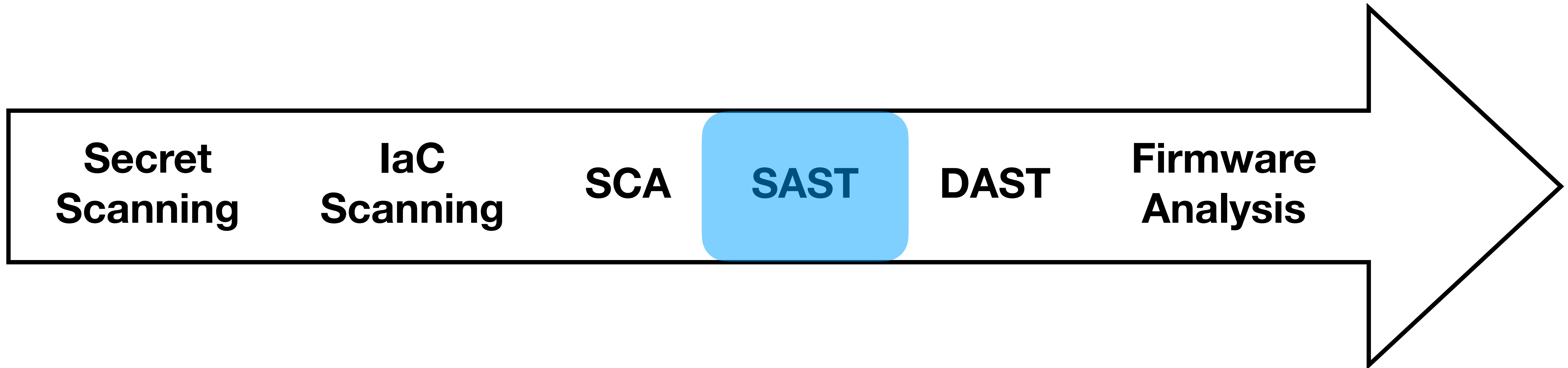
Syft + Grype

```
- uses: aquasecurity/trivy-action@master
with:
  scan-type: 'fs'
  severity: 'HIGH,CRITICAL'
```

Trivy

A Security-Focused CI/CD Pipeline

Overview



Static Application Security Testing (SAST)

detect (security) vulnerabilities without executing the application

“shift left”

usually based on **program querying** / **pattern matching** on the AST level

Static Application Security Testing (SAST)

detect (security) vulnerabilities without executing the application

“shift left”

usually based on **program querying** / **pattern matching** on the AST level



make use of **standard sets of rules** / **patterns** to look for

Static Application Security Testing (SAST)

detect (security) vulnerabilities without executing the application

“shift left”

usually based on **program querying** / **pattern matching** on the AST level



built-in support in GitHub



make use of **standard sets of rules** / **patterns** to look for

Static Application Security Testing (SAST)

detect (security) vulnerabilities without executing the application

“shift left”

usually based on **program querying** / **pattern matching** on the AST level



built-in support in GitHub



make use of **standard sets of rules** / **patterns** to look for

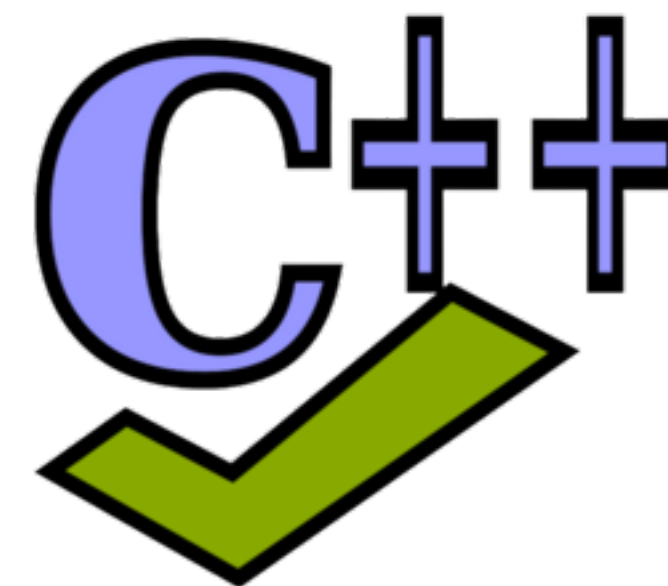
*Interested in setting up more advanced / custom rules?
Let us know!*

Static Application Security Testing (SAST)

detect (security) vulnerabilities without executing the application

“shift left”

usually based on **program querying** / **pattern matching** on the AST level



Adding SAST to the Pipeline

Example: Semgrep

```
sast-semgrep:  
  name: SAST (Semgrep)  
  runs-on: ubuntu-latest  
  container:  
    image: semgrep/semgrep  
  steps:  
    - name: Checkout code  
      uses: actions/checkout@v4  
    - name: Run Semgrep (and output SARIF)  
      run: |  
        semgrep scan \  
          --config p/c \  
          --config p/security-audit \  
          --sarif \  
          -o semgrep.sarif  
    - name: upload Semgrep report (as artifact)  
      uses: actions/upload-artifact@v6  
      with:  
        name: semgrep.sarif  
        path: semgrep.sarif  
    - name: Upload Semgrep SARIF report (as SARIF)  
      uses: github/codeql-action/upload-sarif@v4  
      with:  
        sarif_file: semgrep.sarif  
        category: Semgrep  
  env:  
    GITHUB_TOKEN: ${ secrets.GITHUB_TOKEN }
```

Adding SAST to the Pipeline

Example: Semgrep



```
sast-semgrep:
  name: SAST (Semgrep)
  runs-on: ubuntu-latest
  container:
    image: semgrep/semgrep
  steps:
    - name: Checkout code
      uses: actions/checkout@v4
    - name: Run Semgrep (and output SARIF)
      run: |
        semgrep scan \
          --config p/c \
          --config p/security-audit \
          --sarif \
          -o semgrep.sarif
    - name: upload Semgrep report (as artifact)
      uses: actions/upload-artifact@v6
      with:
        name: semgrep.sarif
        path: semgrep.sarif
    - name: Upload Semgrep SARIF report (as SARIF)
      uses: github/codeql-action/upload-sarif@v4
      with:
        sarif_file: semgrep.sarif
        category: Semgrep
  env:
    GITHUB_TOKEN: ${ secrets.GITHUB_TOKEN }
```

Adding SAST to the Pipeline

Example: Semgrep



```
sast-semgrep:
  name: SAST (Semgrep)
  runs-on: ubuntu-latest
  container:
    image: semgrep/semgrep
  steps:
    - name: Checkout code
      uses: actions/checkout@v4
    - name: Run Semgrep (and output SARIF)
      run: |
        semgrep scan \
          --config p/c \
          --config p/security-audit \
          --sarif \
          -o semgrep.sarif
    - name: upload Semgrep report (as artifact)
      uses: actions/upload-artifact@v6
      with:
        name: semgrep.sarif
        path: semgrep.sarif
    - name: Upload Semgrep SARIF report (as SARIF)
      uses: github/codeql-action/upload-sarif@v4
      with:
        sarif_file: semgrep.sarif
        category: Semgrep
  env:
    GITHUB_TOKEN: ${ secrets.GITHUB_TOKEN }
```

job runs in a Docker container

images come from Docker Hub

Adding SAST to the Pipeline

Example: Semgrep



```
sast-semgrep:
  name: SAST (Semgrep)
  runs-on: ubuntu-latest
  container:
    image: semgrep/semgrep
  steps:
    - name: Checkout code
      uses: actions/checkout@v4
    - name: Run Semgrep (and output SARIF)
      run: |
        semgrep scan \
          --config p/c \
          --config p/security-audit \
          --sarif \
          -o semgrep.sarif
    - name: upload Semgrep report (as artifact)
      uses: actions/upload-artifact@v6
      with:
        name: semgrep.sarif
        path: semgrep.sarif
    - name: Upload Semgrep SARIF report (as SARIF)
      uses: github/codeql-action/upload-sarif@v4
      with:
        sarif_file: semgrep.sarif
        category: Semgrep
      env:
        GITHUB_TOKEN: ${ secrets.GITHUB_TOKEN }
```

output as SARIF

Static Analysis Results Interchange Format

feed SARIF results to GitHub

directly integrated in the
GitHub Security tab

Adding SAST to the Pipeline

Example: Semgrep

```
sast-semgrep:  
  name: SAST (Semgrep)
```

The screenshot displays the GitHub 'Security and quality' interface. The top navigation bar includes 'Code', 'Pull requests', 'Agents', 'Actions', 'Projects', 'Wiki', 'Security and quality' (highlighted with a blue box and showing 15 items), 'Insights', and 'Settings'. The left sidebar shows 'Security and quality' with sub-sections: 'Overview', 'Findings' (with 'Dependabot' and 'Code scanning' (15 items) listed), and 'Reporting' (with 'Security policy' and 'Advisories' listed). The main content area is titled 'Code scanning' and features a warning banner: 'CodeQL and Semgrep OSS are reporting warnings. Check the status page for help.' Below this is a search bar with the query 'is:open branch:main'. A summary shows '15 Open' and '0 Closed' findings. A table of findings follows, with columns for checkboxes, severity, description, detection details, and branch. The findings include two 'Critical' 'Code injection' issues, one 'High' 'Checkout of untrusted code in trusted context' issue, and three 'Medium' issues related to unused functions and variables.

Code scanning

⚠️ CodeQL and Semgrep OSS are reporting warnings. Check the [status page](#) for help. Tools 3 + Add tool

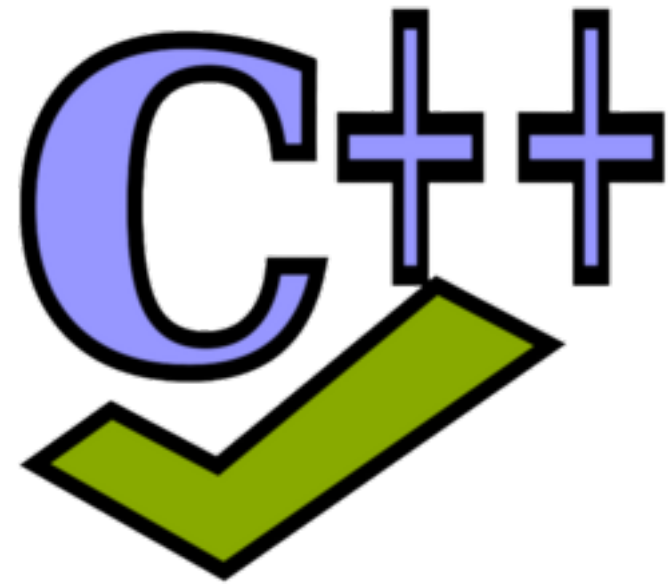
is:open branch:main

15 Open 0 Closed Language Tool Rule Severity Sort

<input type="checkbox"/>	🚨 Code injection Critical	#3 opened 3 weeks ago • Detected by CodeQL in .github/workflows/firmware-build-vulnerabl... :47	main
<input type="checkbox"/>	🚨 Code injection Critical	#2 opened 3 weeks ago • Detected by CodeQL in .github/workflows/firmware-build-vulnerabl... :46	main
<input type="checkbox"/>	🚨 Checkout of untrusted code in trusted context High	#1 opened 3 weeks ago • Detected by CodeQL in .github/workflows/firmware-build-vulnerabl... :30	main
<input type="checkbox"/>	🚨 The function 'loop' is never used. Medium Test	#15 opened 2 weeks ago • Detected by CppCheck in ../../src/main.cpp :408	main
<input type="checkbox"/>	🚨 The function 'setup' is never used. Medium Test	#14 opened 2 weeks ago • Detected by CppCheck in ../../src/main.cpp :372	main
<input type="checkbox"/>	🚨 Variable 'searchStart' is assigned a value that is never used. Medium Test	#13 opened 2 weeks ago • Detected by CppCheck in ../../src/main.cpp :229	main
<input type="checkbox"/>	🚨 Workflow does not contain permissions Medium	#4 opened 3 weeks ago • Detected by CodeQL in .github/workflows/SCA.yml :5	main

Adding SAST to the Pipeline

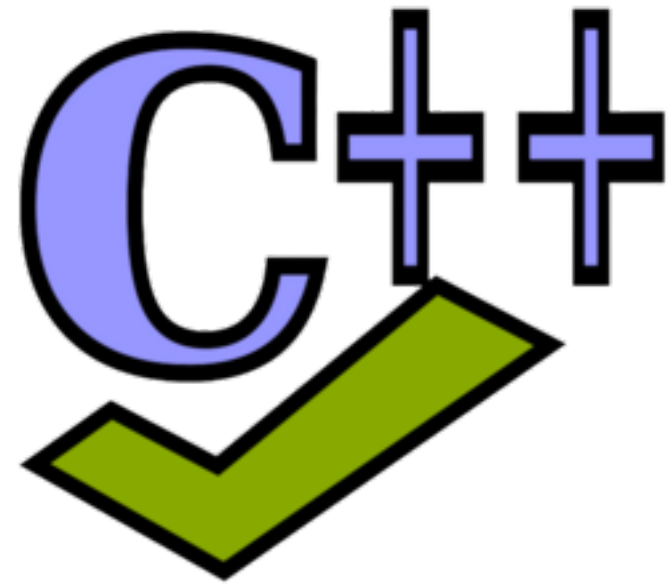
Example: cppcheck



```
sast-cppcheck:
  name: SAST (cppcheck)
  runs-on: ubuntu-latest
  steps:
    - name: Checkout code
      uses: actions/checkout@v4
    - name: Restore cppcheck cache
      id: cache
      uses: actions/cache@v4
      with:
        path: ~/.local
        key: cppcheck-${{ runner.os }}-v1
    - name: Build cppcheck from source
      if: steps.cache.outputs.cache-hit != 'true'
      run: |
        sudo apt update && sudo apt install -y build-essential cmake libpcre3-dev libtinyxml2-dev git
        git clone https://github.com/cppcheck-opensource/cppcheck.git
        cd cppcheck
        mkdir build && cd build
        cmake -DCMAKE_INSTALL_PREFIX=$HOME/.local ..
        make -j$(nproc)
        sudo make install
        echo "$HOME/.local/bin" >> $GITHUB_PATH
    - name: Run cppcheck
      run: |
        cppcheck --enable=all --output-format=sarif --output-file=cppcheck.sarif ../test-it/src
    - name: Upload cppcheck report (as artifact)
      uses: actions/upload-artifact@v6
      with:
        name: cppcheck.sarif
        path: cppcheck.sarif
    - name: Upload cppcheck report (as SARIF)
      uses: github/codeql-action/upload-sarif@v4
      with:
        sarif_file: cppcheck.sarif
        category: cppcheck
  env:
    GITHUB_TOKEN: ${ secrets.GITHUB_TOKEN }
```

Adding SAST to the Pipeline

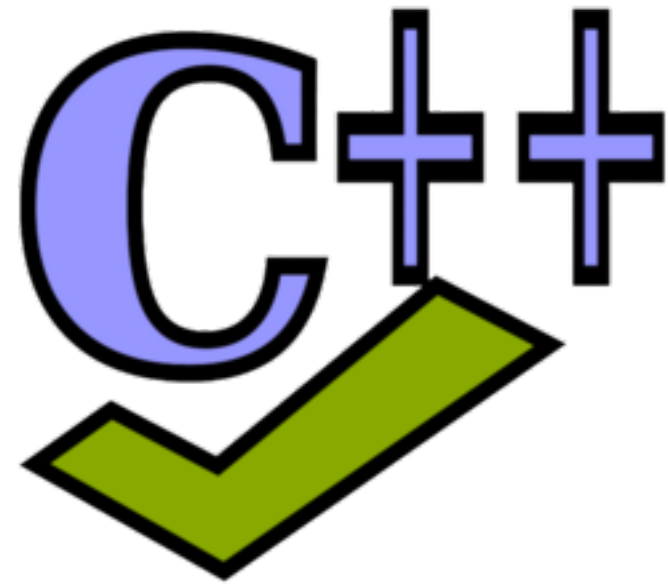
Example: cppcheck



```
sast-cppcheck:
  name: SAST (cppcheck)
  runs-on: ubuntu-latest
  steps:
    - name: Checkout code
      uses: actions/checkout@v4
    - name: Restore cppcheck cache
      id: cache
      uses: actions/cache@v4
      with:
        path: ~/.local
        key: cppcheck-${{ runner.os }}-v1
    - name: Build cppcheck from source
      if: steps.cache.outputs.cache-hit != 'true'
      run: |
        sudo apt update && sudo apt install -y build-essential cmake libpcre3-dev libtinyxml2-dev git
        git clone https://github.com/cppcheck-opensource/cppcheck.git
        cd cppcheck
        mkdir build && cd build
        cmake -DCMAKE_INSTALL_PREFIX=$HOME/.local ..
        make -j$(nproc)
        sudo make install
        echo "$HOME/.local/bin" >> $GITHUB_PATH
    - name: Run cppcheck
      run: |
        cppcheck --enable=all --output-format=sarif --output-file=cppcheck.sarif ../test-it/src
    - name: Upload cppcheck report (as artifact)
      uses: actions/upload-artifact@v6
      with:
        name: cppcheck.sarif
        path: cppcheck.sarif
    - name: Upload cppcheck report (as SARIF)
      uses: github/codeql-action/upload-sarif@v4
      with:
        sarif_file: cppcheck.sarif
        category: cppcheck
  env:
    GITHUB_TOKEN: ${ secrets.GITHUB_TOKEN }
```

Adding SAST to the Pipeline

Example: cppcheck

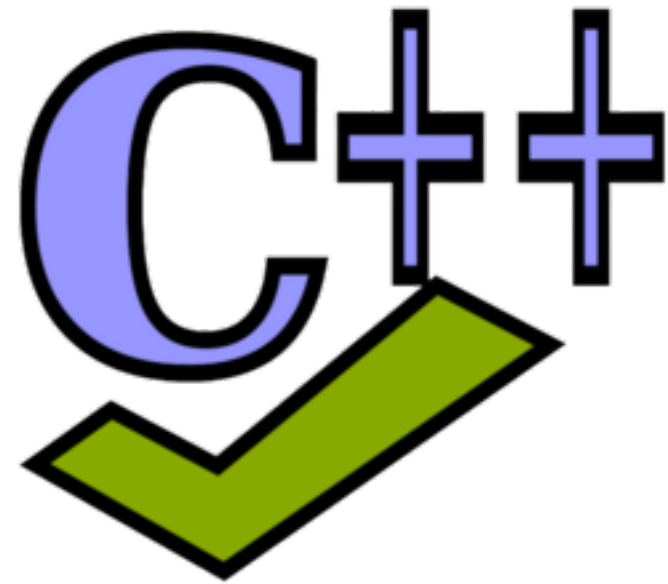


```
sast-cppcheck:
  name: SAST (cppcheck)
  runs-on: ubuntu-latest
  steps:
    - name: Checkout code
      uses: actions/checkout@v4
    - name: Restore cppcheck cache
      id: cache
      uses: actions/cache@v4
      with:
        path: ~/.local
        key: cppcheck-${{ runner.os }}-v1
    - name: Build cppcheck from source
      if: steps.cache.outputs.cache-hit != 'true'
      run: |
        sudo apt update && sudo apt install -y build-essential cmake libpcre3-dev libtinyxml2-dev git
        git clone https://github.com/cppcheck-opensource/cppcheck.git
        cd cppcheck
        mkdir build && cd build
        cmake -DCMAKE_INSTALL_PREFIX=$HOME/.local ..
        make -j$(nproc)
        sudo make install
        echo "$HOME/.local/bin" >> $GITHUB_PATH
    - name: Run cppcheck
      run: |
        cppcheck --enable=all --output-format=sarif --output-file=cppcheck.sarif ../test-it/src
    - name: Upload cppcheck report (as artifact)
      uses: actions/upload-artifact@v6
      with:
        name: cppcheck.sarif
        path: cppcheck.sarif
    - name: Upload cppcheck report (as SARIF)
      uses: github/codeql-action/upload-sarif@v4
      with:
        sarif_file: cppcheck.sarif
        category: cppcheck
  env:
    GITHUB_TOKEN: ${ secrets.GITHUB_TOKEN }
```

caching to avoid (costly) dependency re-installations

Adding SAST to the Pipeline

Example: cppcheck



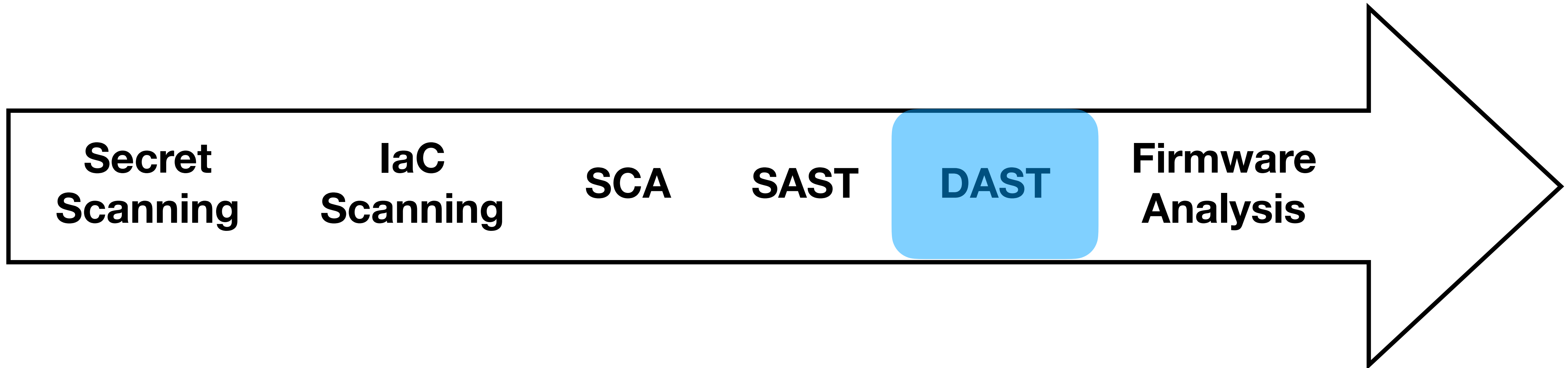
```
sast-cppcheck:
  name: SAST (cppcheck)
  runs-on: ubuntu-latest
  steps:
    - name: Checkout code
      uses: actions/checkout@v4
    - name: Restore cppcheck cache
      id: cache
      uses: actions/cache@v4
      with:
        path: ~/.local
        key: cppcheck-${ runner.os }-v1
    - name: Build cppcheck from source
      if: steps.cache.outputs.cache-hit != 'true'
      run: |
        sudo apt update && sudo apt install -y build-essential cmake libpcre3-dev libtinyxml2-dev git
        git clone https://github.com/cppcheck-opensource/cppcheck.git
        cd cppcheck
        mkdir build && cd build
        cmake -DCMAKE_INSTALL_PREFIX=$HOME/.local ..
        make -j$(nproc)
        sudo make install
        echo "$HOME/.local/bin" >> $GITHUB_PATH
    - name: Run cppcheck
      run: |
        cppcheck --enable=all --output-format=sarif --output-file=cppcheck.sarif ../test-it/src
    - name: Upload cppcheck report (as artifact)
      uses: actions/upload-artifact@v6
      with:
        name: cppcheck.sarif
        path: cppcheck.sarif
    - name: Upload cppcheck report (as SARIF)
      uses: github/codeql-action/upload-sarif@v4
      with:
        sarif_file: cppcheck.sarif
        category: cppcheck
      env:
        GITHUB_TOKEN: ${ secrets.GITHUB_TOKEN }
```

output as SARIF

upload to GitHub Security

A Security-Focused CI/CD Pipeline

Overview



Dynamic Application Security Testing (DAST)

run the application with **unexpected / random inputs** to find vulnerabilities

fuzzing!

Dynamic Application Security Testing (DAST)

run the application with **unexpected / random inputs** to find vulnerabilities

fuzzing!

usually run on nightly or pre-release tests, not per commit (costly)

Dynamic Application Security Testing (DAST)

run the application with **unexpected / random inputs** to find vulnerabilities

fuzzing!

usually run on nightly or pre-release tests, not per commit (costly)

➔ protocol fuzzing



➔ input fuzzing



➔ web/API fuzzing



A Security-Focused CI/CD Pipeline

Overview



Firmware Analysis

source-code level and **binary analysis**

Firmware Analysis

source-code level and **binary analysis**

EMBA

README Code of conduct Contributing GPL-3.0 license Security



ShellCheck no status Made with Bash License GPL-3.0 contributors 25 Stars 3.5k Forks 303 docker pulls 176k

Tweet

EMBA

The security analyzer for firmware of embedded devices

EMBA is designed as the central firmware analysis and SBOM tool for penetration testers, product security teams, developers and responsible product managers. It supports the complete security analysis process starting with *firmware extraction*, doing *static analysis* and *dynamic analysis* via emulation, building the SBOM and finally generating a web based vulnerability report. *EMBA* automatically discovers possible weak spots and vulnerabilities in firmware. Examples are insecure binaries, old and outdated software components, potentially vulnerable scripts, or hard-coded passwords. *EMBA* is a command line tool with the possibility to generate an easy-to-use web report for further analysis

Firmware Analysis

source-code level and **binary analysis**

README Code of conduct Contributing GPL-3.0 license Security

EMBA

ShellCheck no status Made with Bash License GPL-3.0 contributors 25 Stars 3.5k Forks 303 docker pulls 176k

Tweet

EMBA

The security analyzer for firmware of embedded devices

EMBA is designed as the central firmware analysis and SBOM tool for penetration testers, product security teams, developers and responsible product managers. It supports the complete security analysis process starting with *firmware extraction*, doing *static analysis* and *dynamic analysis* via emulation, building the SBOM and finally generating a web based vulnerability report. *EMBA* automatically discovers possible weak spots and vulnerabilities in firmware. Examples are insecure binaries, old and outdated software components, potentially vulnerable scripts, or hard-coded passwords. *EMBA* is a command line tool with the possibility to generate an easy-to-use web report for further analysis

EMBA

*Interested? Let us know
for future workshops!*

A Security-Focused CI/CD Pipeline

Overview



A Security-Focused CI/CD Pipeline

Overview

