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

<u>PART 1</u>:
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 / **C**ontinuous **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

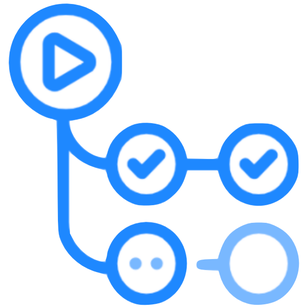
Continuous **I**ntegration / **C**ontinuous **D**elivery

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

“reproducibility”: automated builds

Jenkins

CI/CD



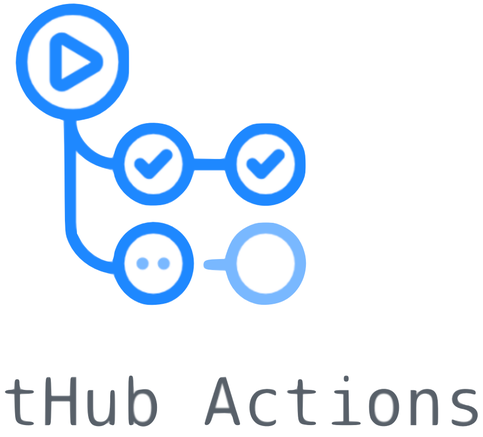
GitHub Actions

Setting Up a CI/CD Pipeline

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

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

“reproducibility”: automated builds

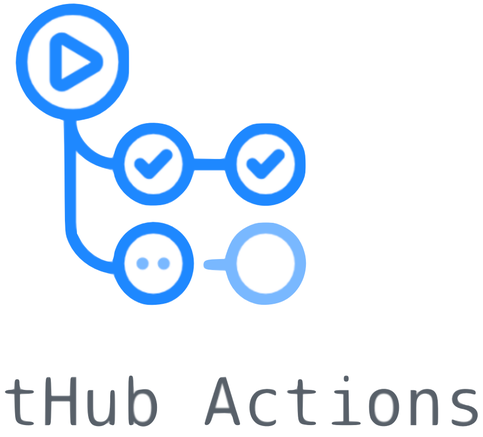


Setting Up a CI/CD Pipeline

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

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

“reproducibility”: automated builds



Migrating?

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

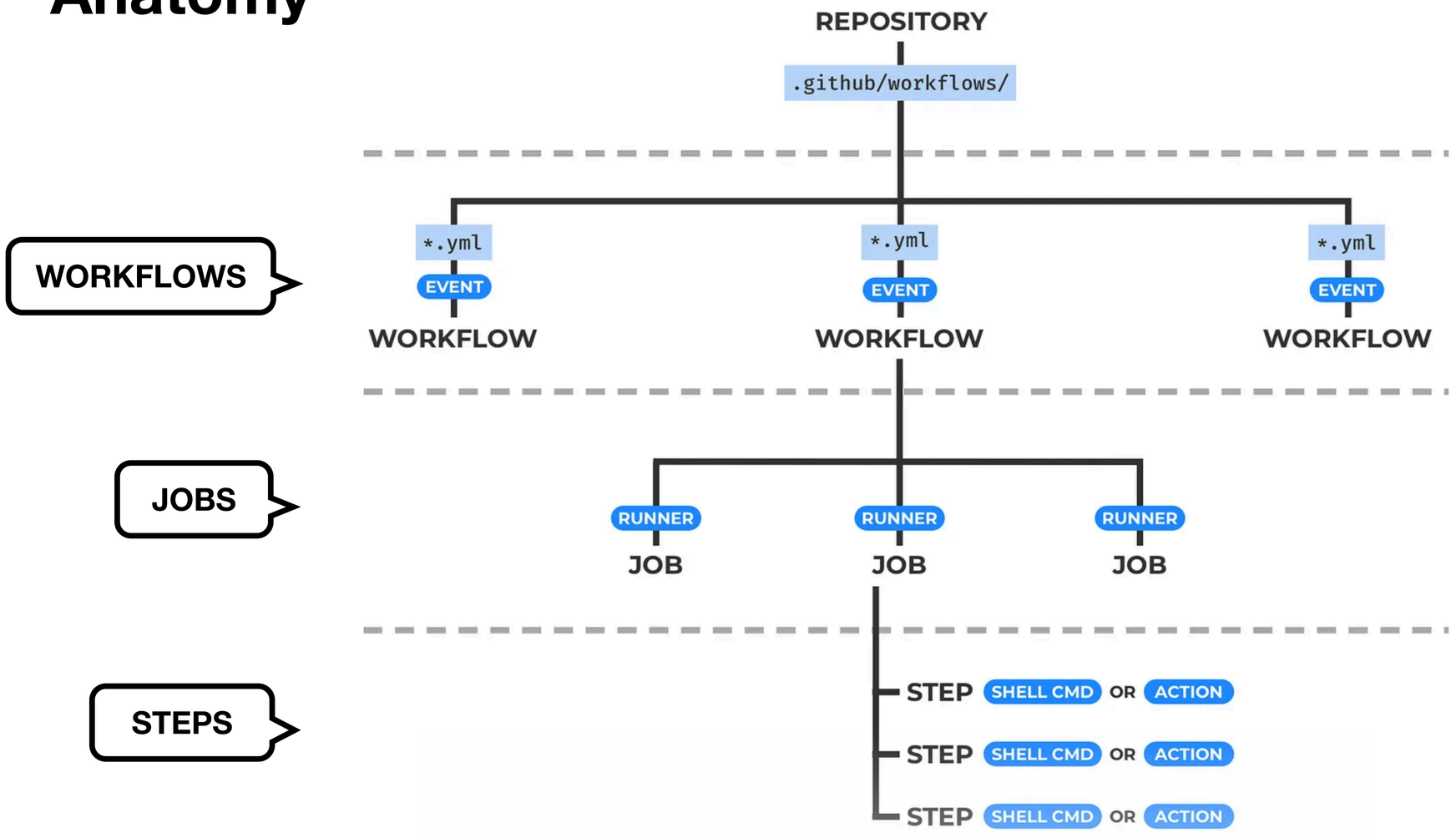
What's in a Pipeline?

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

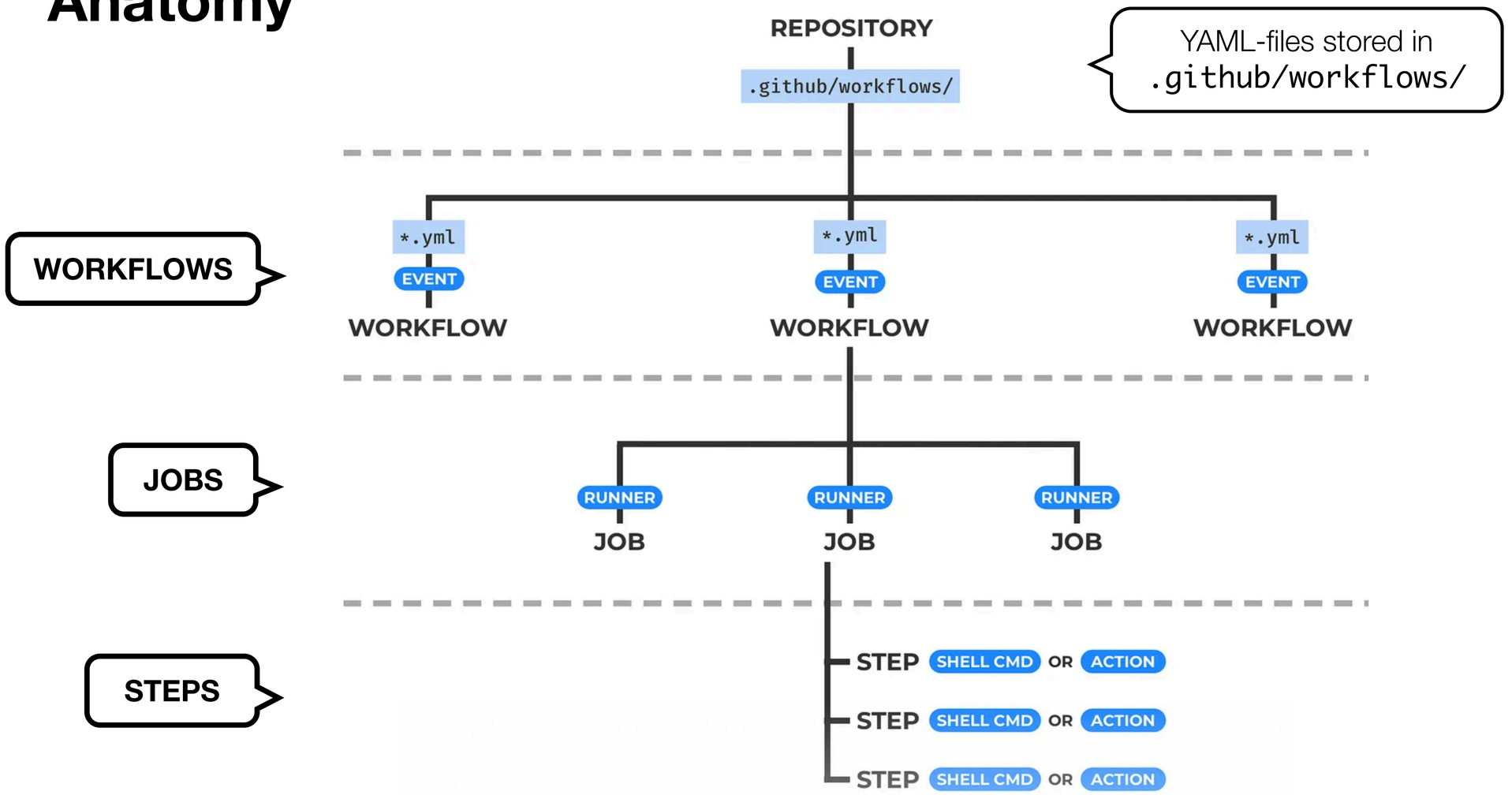
Anatomy



(image source: https://itnext.io/getting-started-with-github-actions-fe94167dbc6d)

GitHub Actions: A Primer

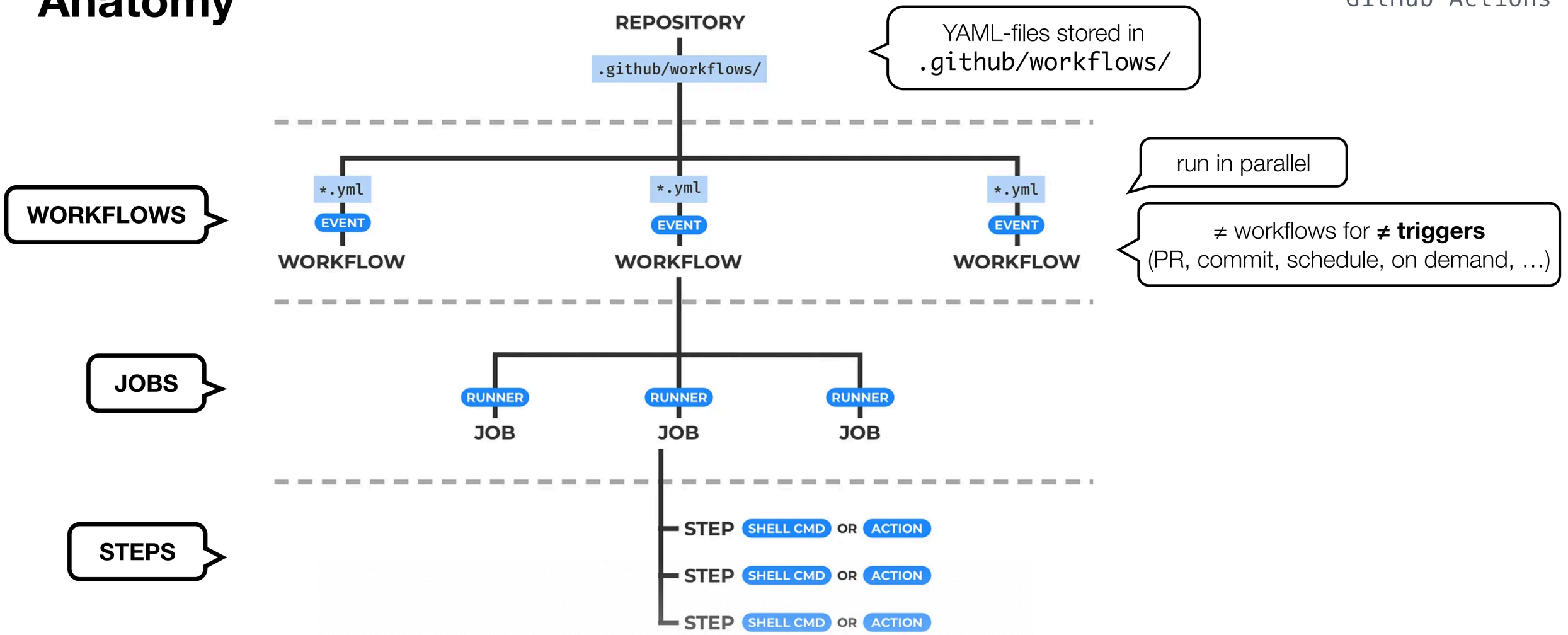
Anatomy



(image source: https://itnext.io/getting-started-with-github-actions-fe94167dbc6d)

GitHub Actions: A Primer

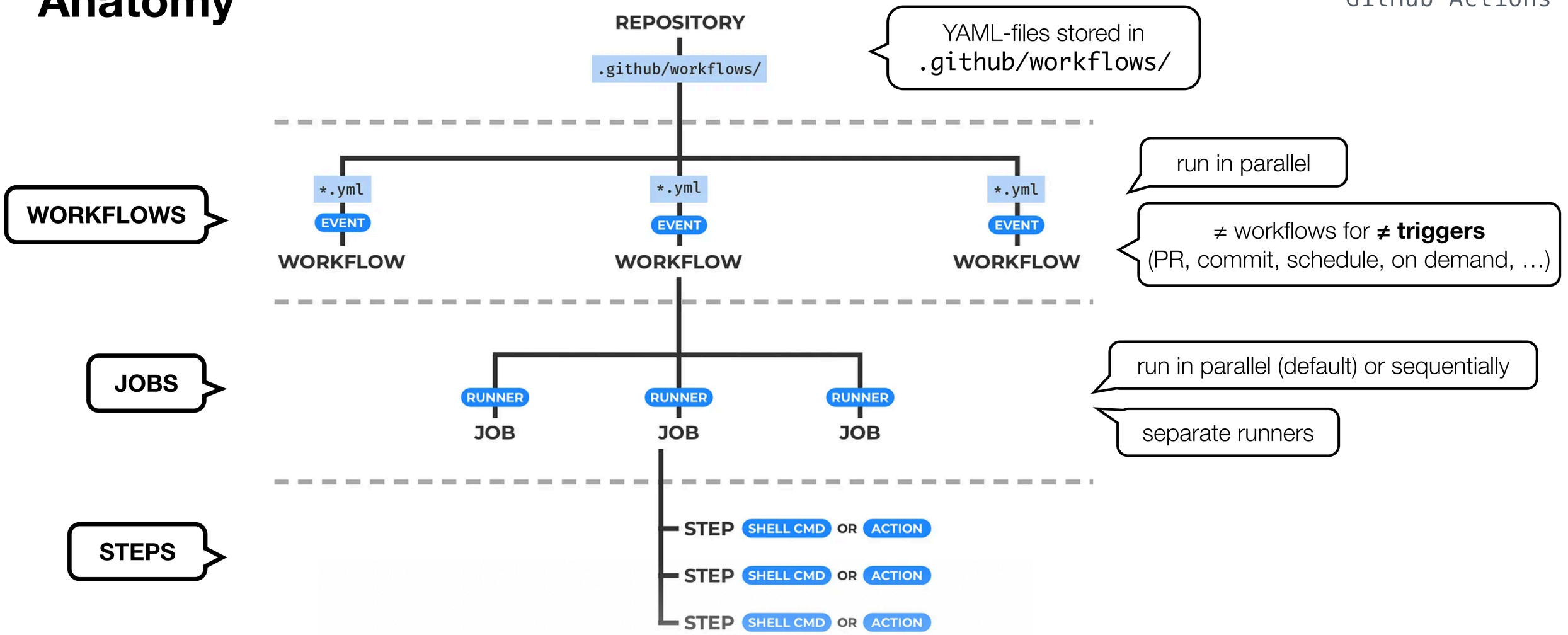
Anatomy



(image source: https://itnext.io/getting-started-with-github-actions-fe94167dbc6d)

GitHub Actions: A Primer

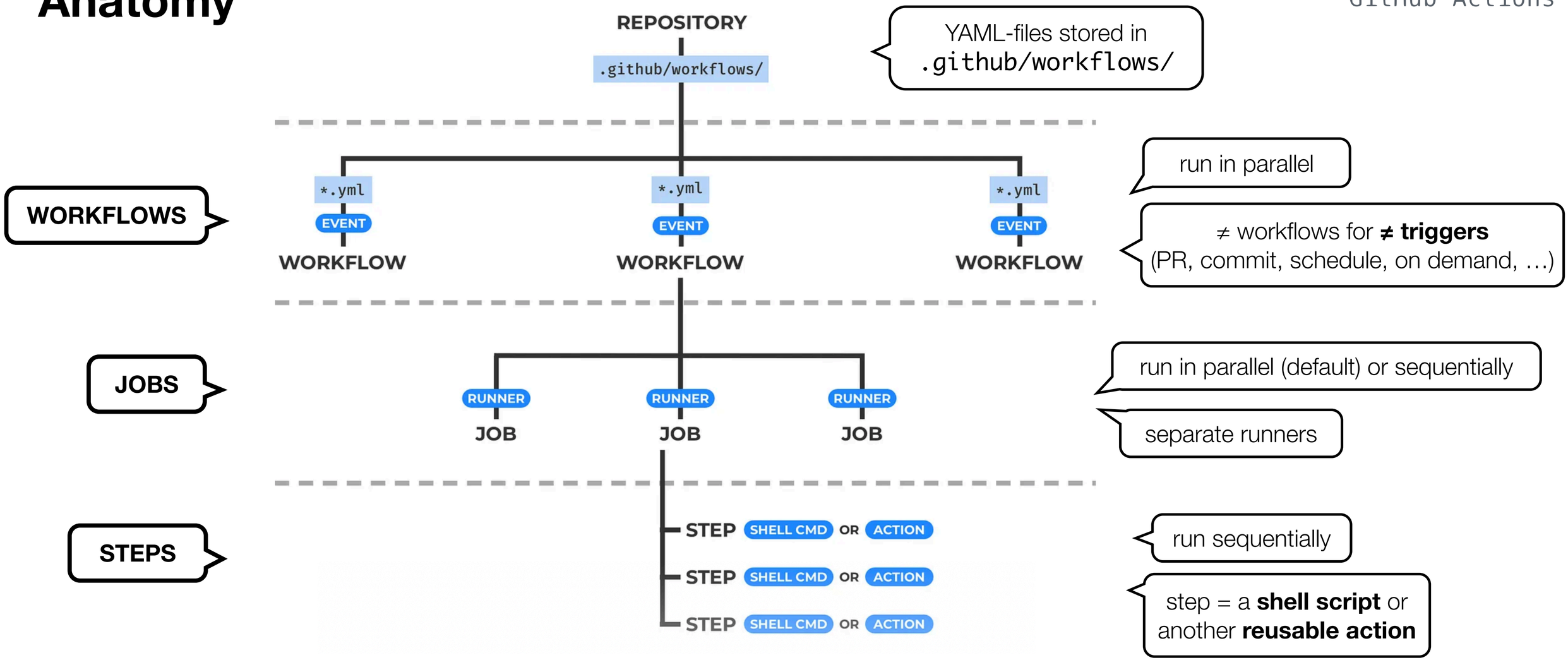
Anatomy



(image source: https://itnext.io/getting-started-with-github-actions-fe94167dbc6d)

GitHub Actions: A Primer

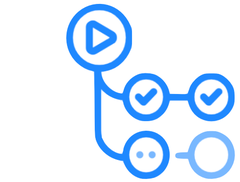
Anatomy

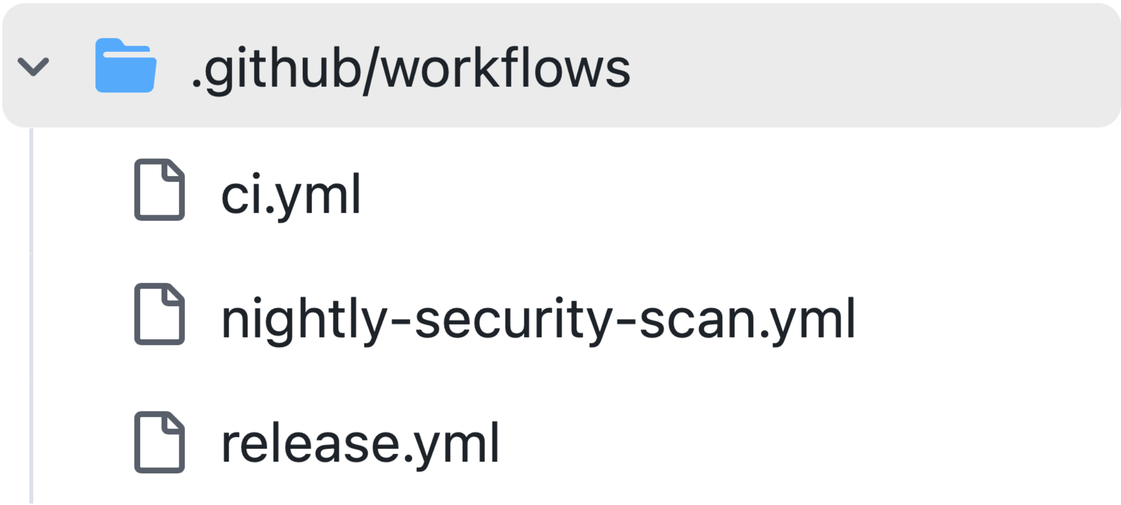


(image source: https://itnext.io/getting-started-with-github-actions-fe94167dbc6d)

GitHub Actions: A Primer

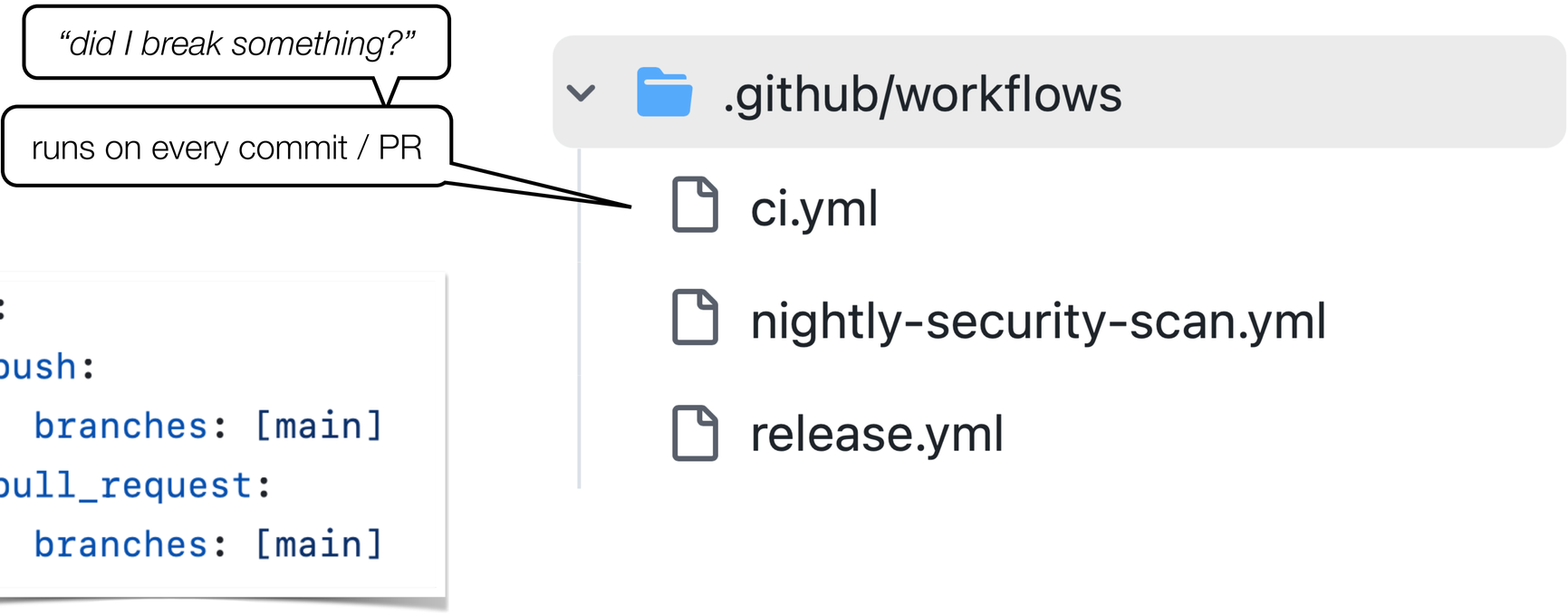
Example Repository





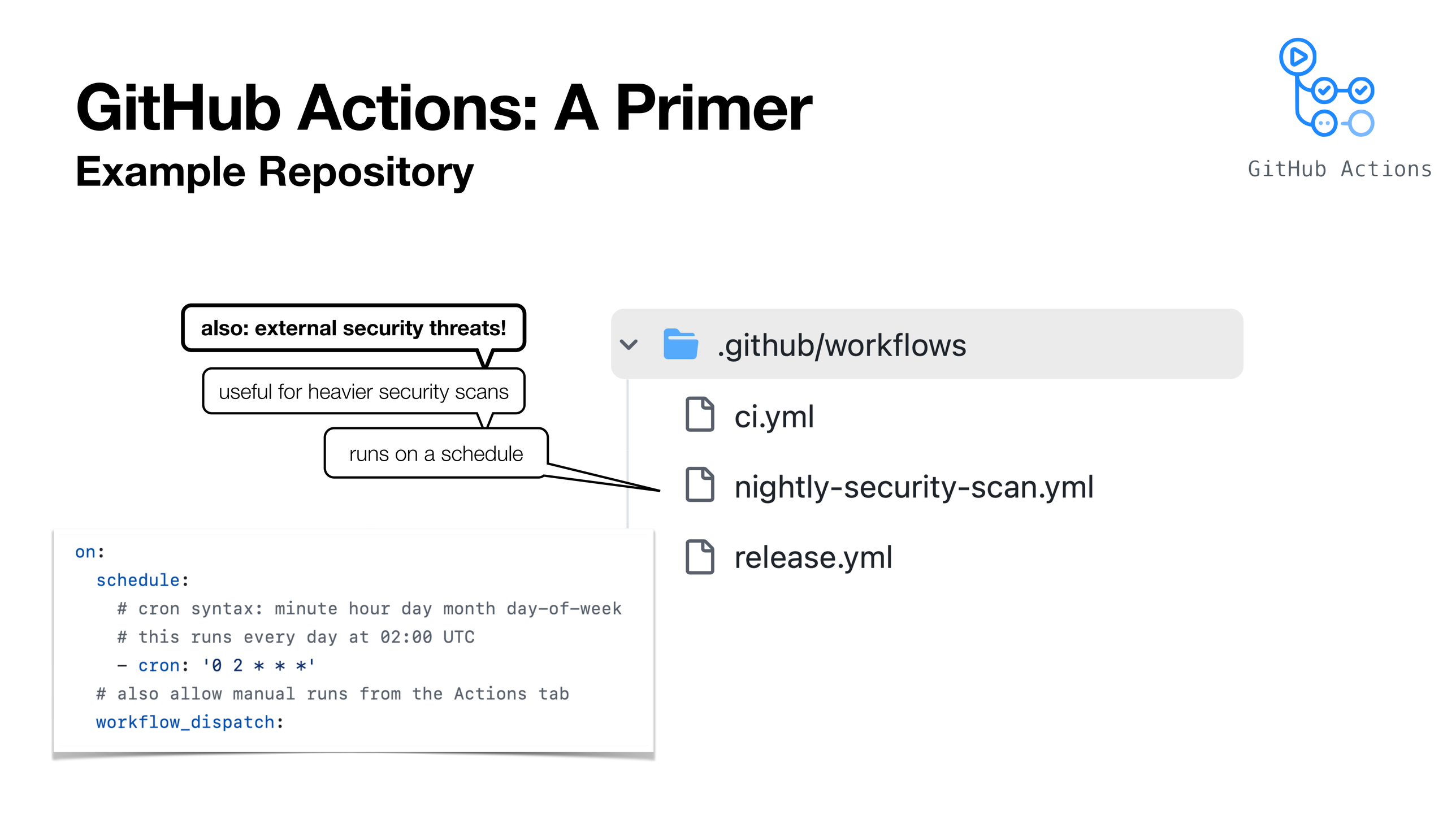
GitHub Actions: A Primer

Example Repository



GitHub Actions: A Primer

Example Repository




```yaml
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:
```

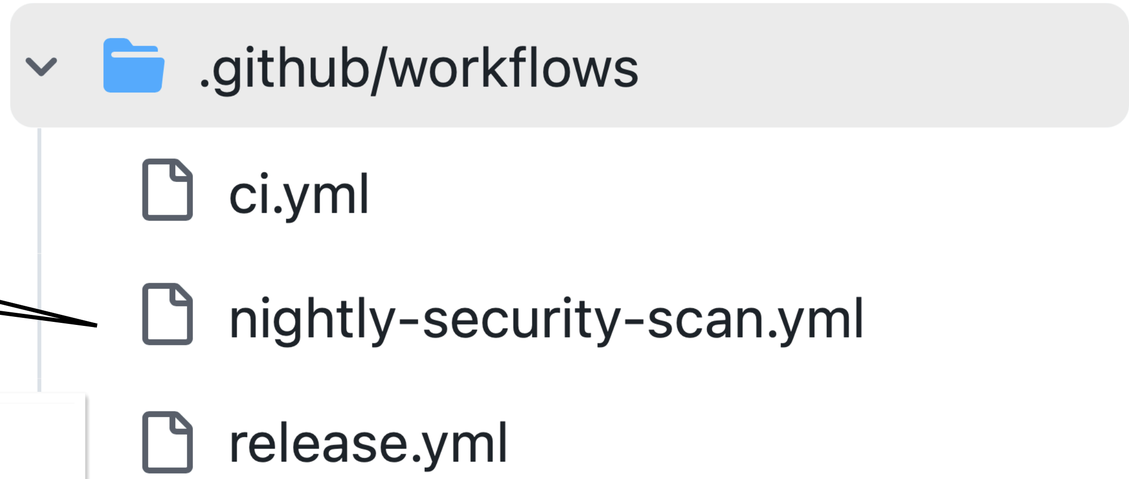
GitHub Actions: A Primer

Example Repository

also: external security threats!

useful for heavier security scans

runs on a schedule

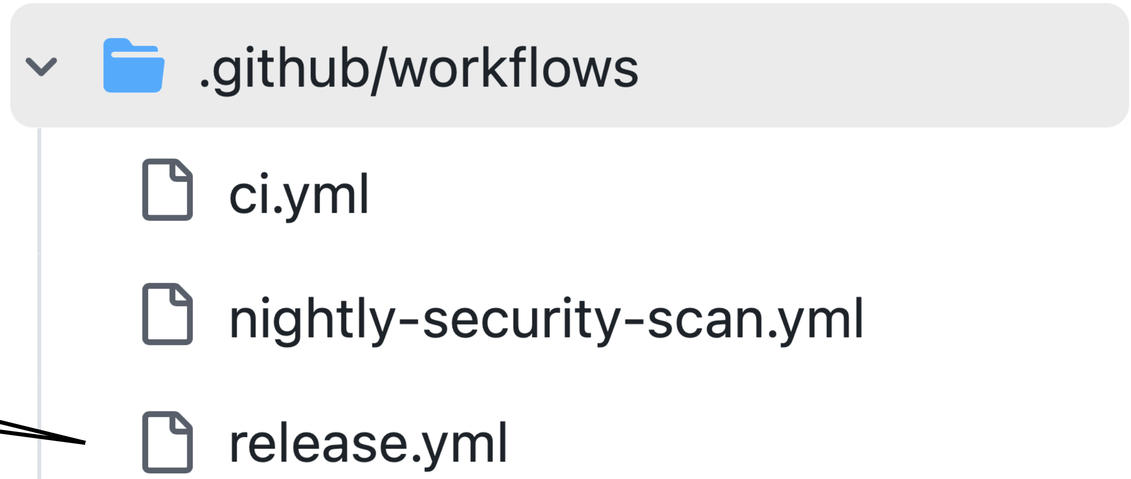



```yaml
on:
  schedule:
    # cron syntax: minute hour day month day-of-week
    # this runs every day at 02:00 UTC
    - cron: '0 2 * * *'
  # also allow manual runs from the Actions tab
  workflow_dispatch:
```

can also be run manually

GitHub Actions: A Primer

Example Repository



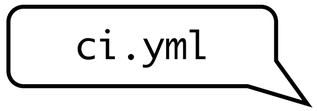
SBOM generation, signing, …

runs when a new tag is pushed

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

GitHub Actions: A Primer

Example Workflow

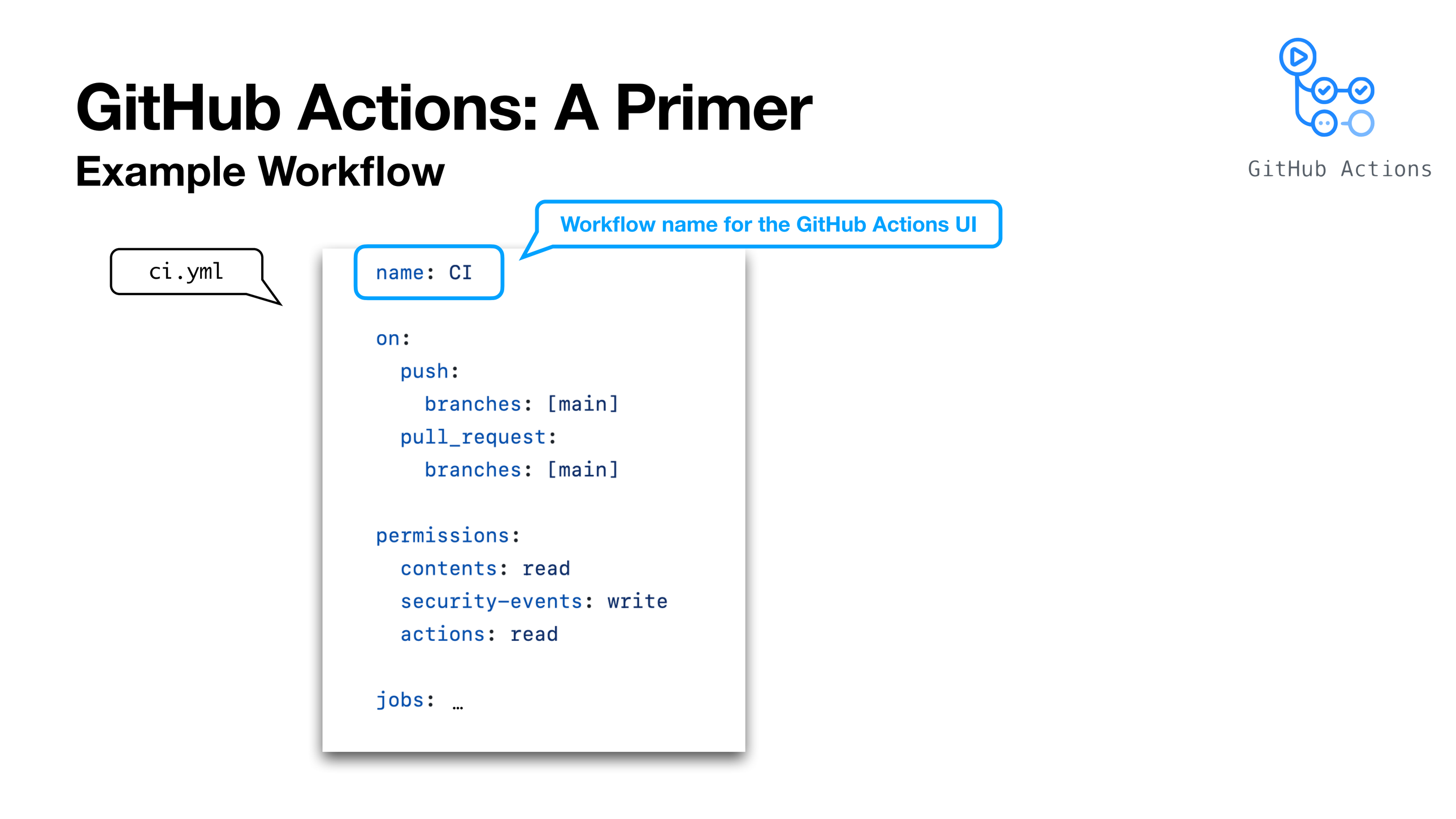



```yaml
name: CI

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

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

jobs: …
```

GitHub Actions

GitHub Actions: A Primer

Example Workflow



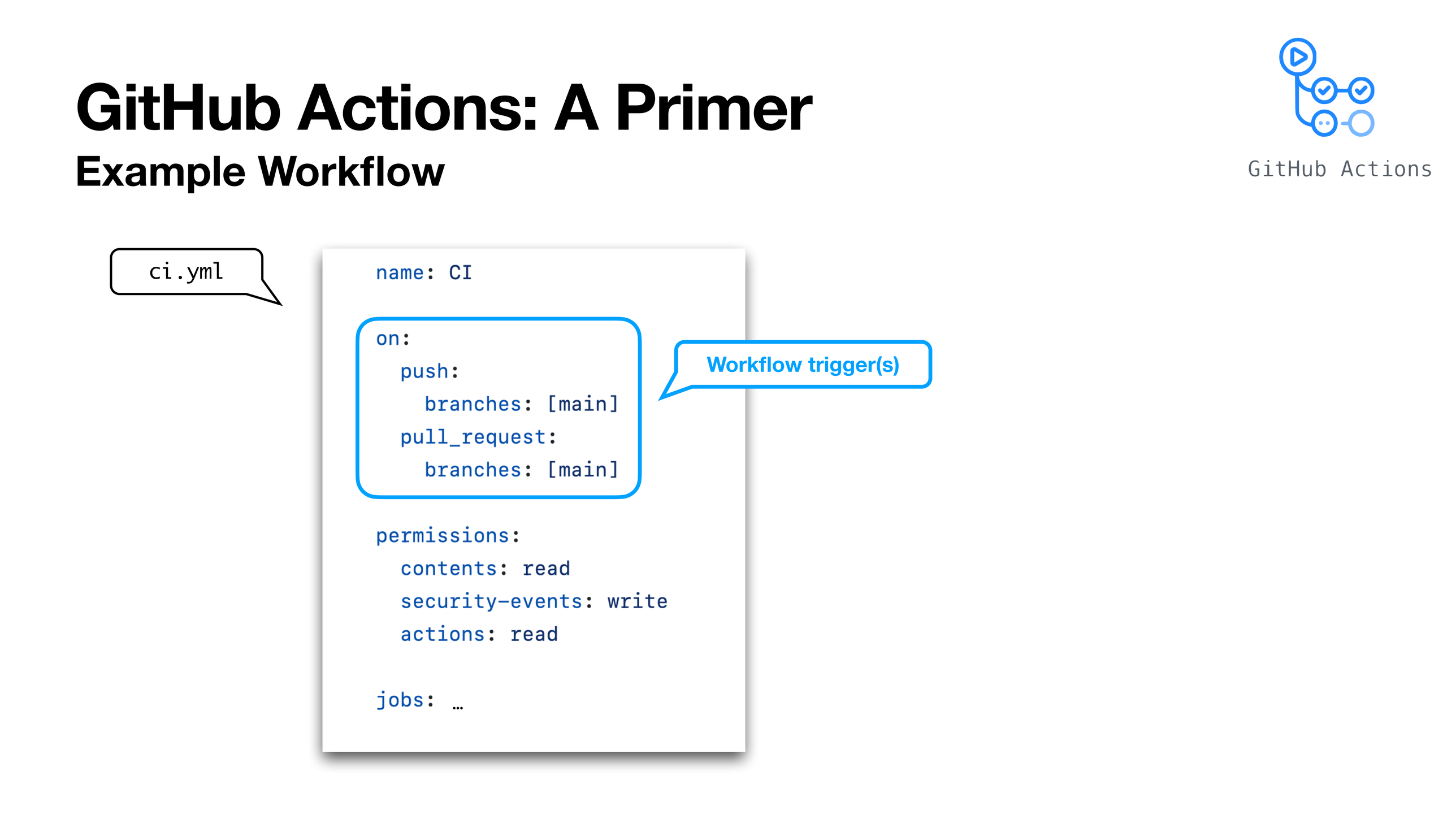
ci.yml

Workflow name for the GitHub Actions UI

```yaml
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



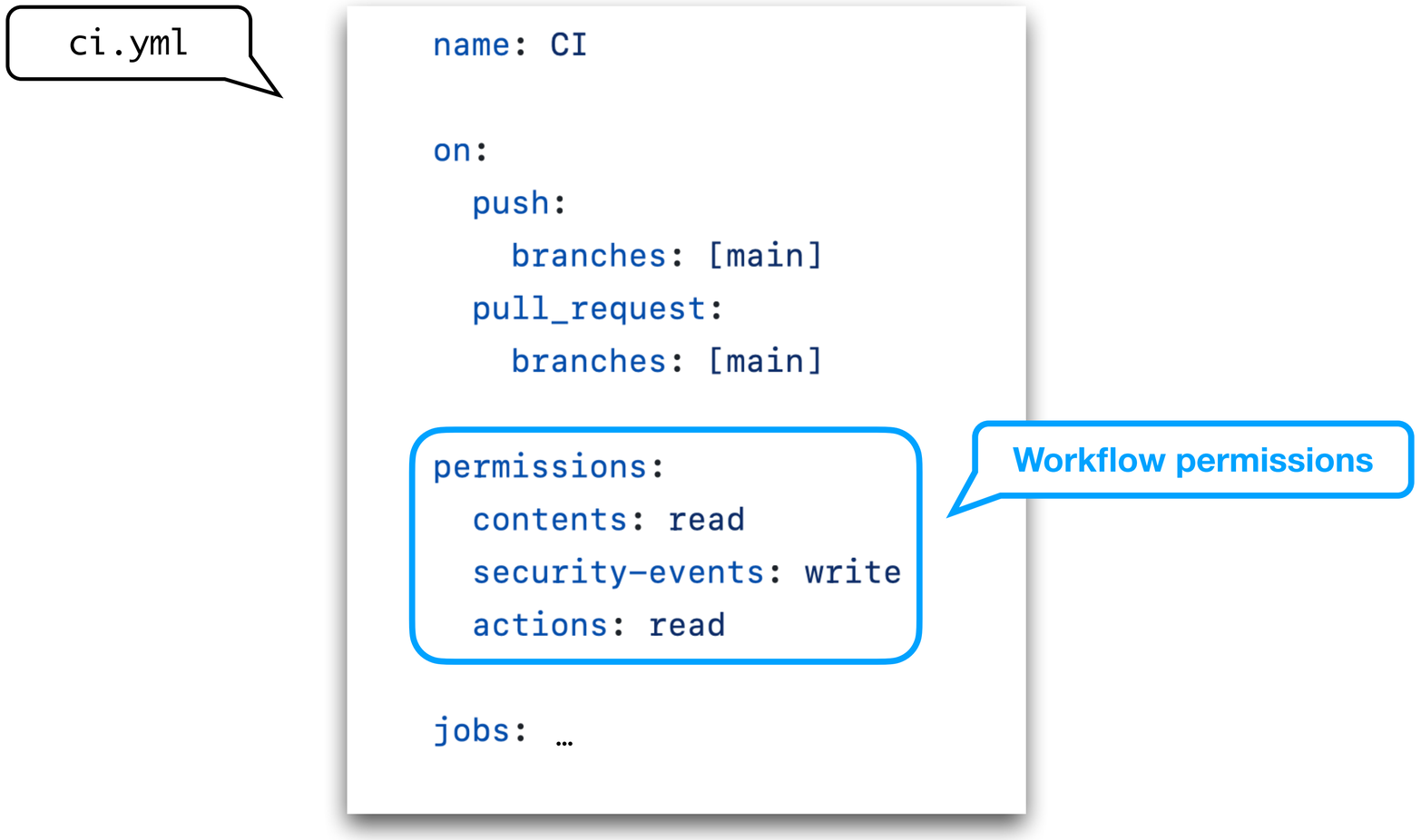
ci.yml

```yaml
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

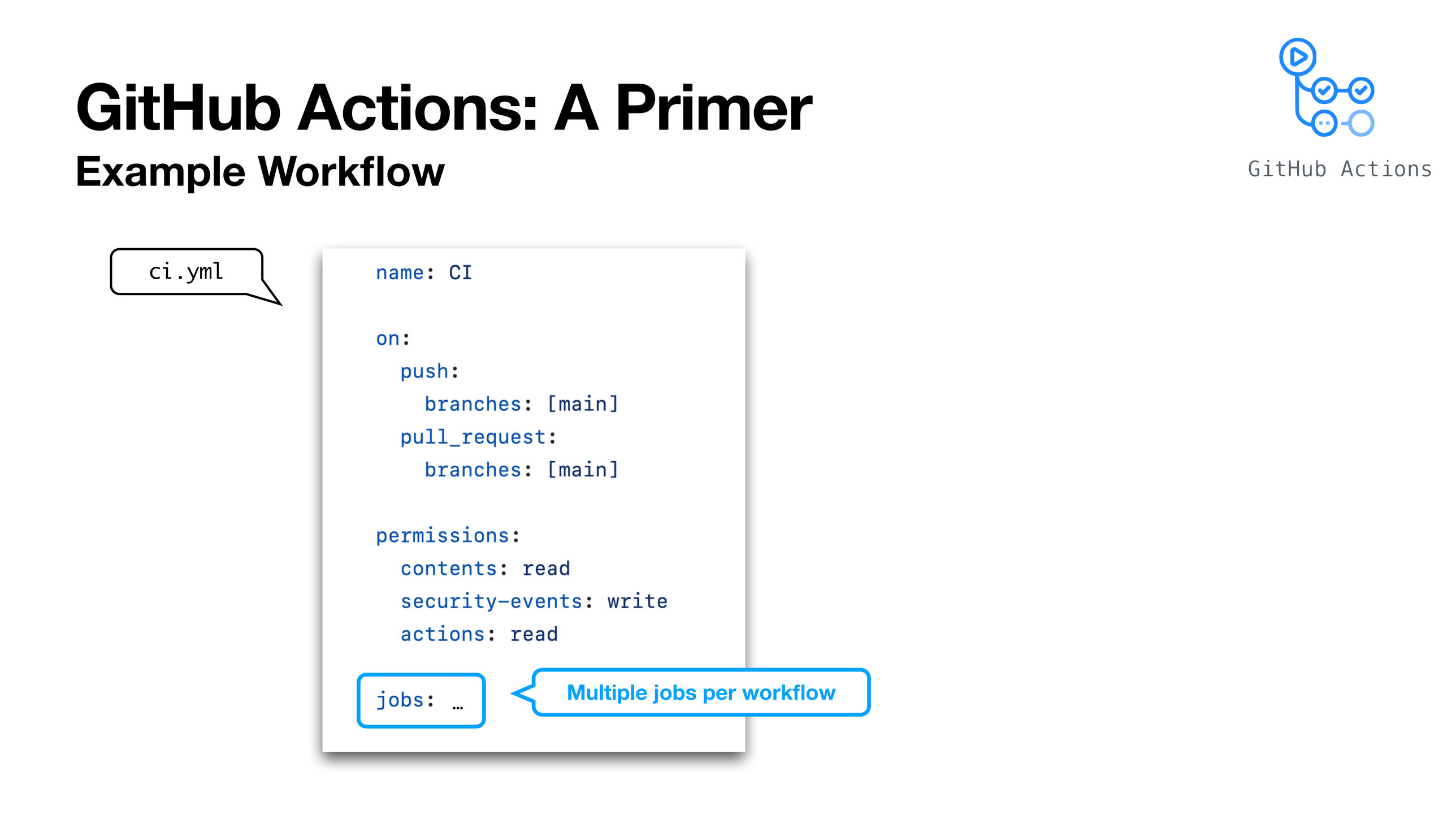
Example Workflow



GitHub Actions

GitHub Actions: A Primer

Example Workflow



ci.yml

```yaml
name: CI

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

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

jobs: …
```

Multiple jobs per workflow

GitHub A ner

Example Wo

ci.yml

```yaml
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: ...
```

GitHub Actions

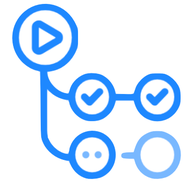
GitHub A ner

Example Wo

ci.yml

```yaml
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 A ner

Example Wo

GitHub Actions

ci.yml

```yaml
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: ...
```

each job has its own runner (provided by GitHub)

GitHub A ner

Example Wo

ci.yml

```yaml
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: ...
```

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 A ner

Example Wo

ci.yml

```yaml
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: ...
```

GitHub Actions

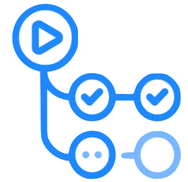
By default, all jobs run in parallel …

GitHub A ner

Example Wo

ci.yml

```yaml
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: ...
```



By default, all jobs run in parallel …

… unless job dependencies are explicitly specified

GitHub A ner

Example Wo

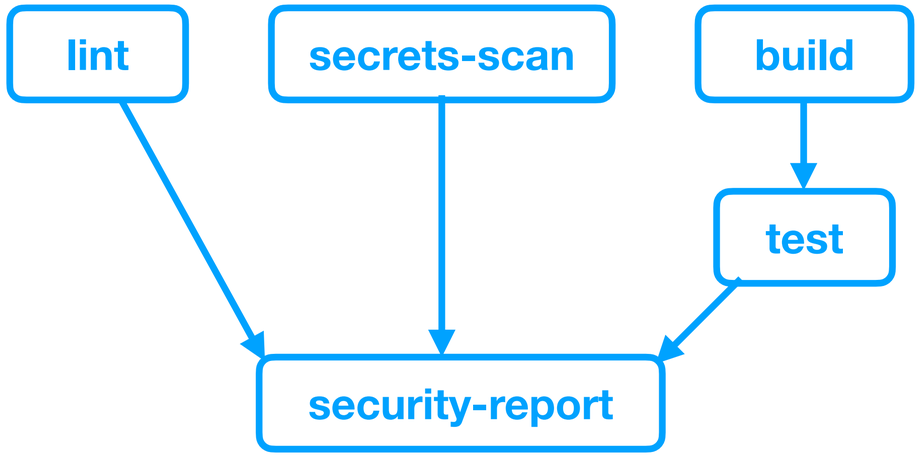
ci.yml

```yaml
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: ...
```

GitHub Actions

By default, all jobs run in parallel …

… unless job dependencies are explicitly specified



GitHub A ner

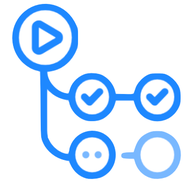
Example Wo

ci.yml

```yaml
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: ...
```

conditional job execution

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



GitHub Actions

GitHub A ner

Example Wo

ci.yml

```yaml
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

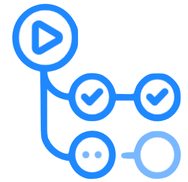
GitHub Actions

GitHub A…er

Example Wo…

ci.yml

```yaml
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: ...
```



GitHub Actions

GitHub A…er

Example Wo…

ci.yml

```yaml
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: ...
```

5 steps in the `build` job

steps run sequentially on the same runner

GitHub Actions

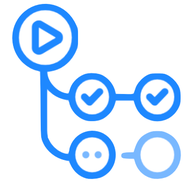
GitHub A…er

Example Wo…

ci.yml

```yaml
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: ...
```

(optional) step name for the GitHub Actions UI



GitHub Actions

GitHub A…er

Example Wo…

ci.yml

```yaml
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: ...
```

GitHub Actions

2 kinds of steps …

GitHub A…er

Example Wo…

ci.yml

```yaml
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: ...
```

GitHub Actions

2 kinds of steps …

scripts (using `run:`)

GitHub A…er

Example Wo…

ci.yml

```yaml
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: ...
```

GitHub Actions

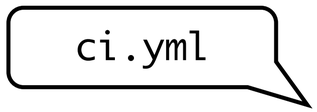
2 kinds of steps …

scripts (using `run:`)

reusable actions (using `uses:`)

GitHub A…er

Example Wo…




```yaml
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

pass arguments

2 kinds of steps …

scripts (using `run:`)

reusable actions (using `uses:`)

GitHub Actions

GitHub A…er

Example Wo…

ci.yml

```yaml
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: ...
```

GitHub Actions

2 kinds of steps …

scripts (using `run:`)

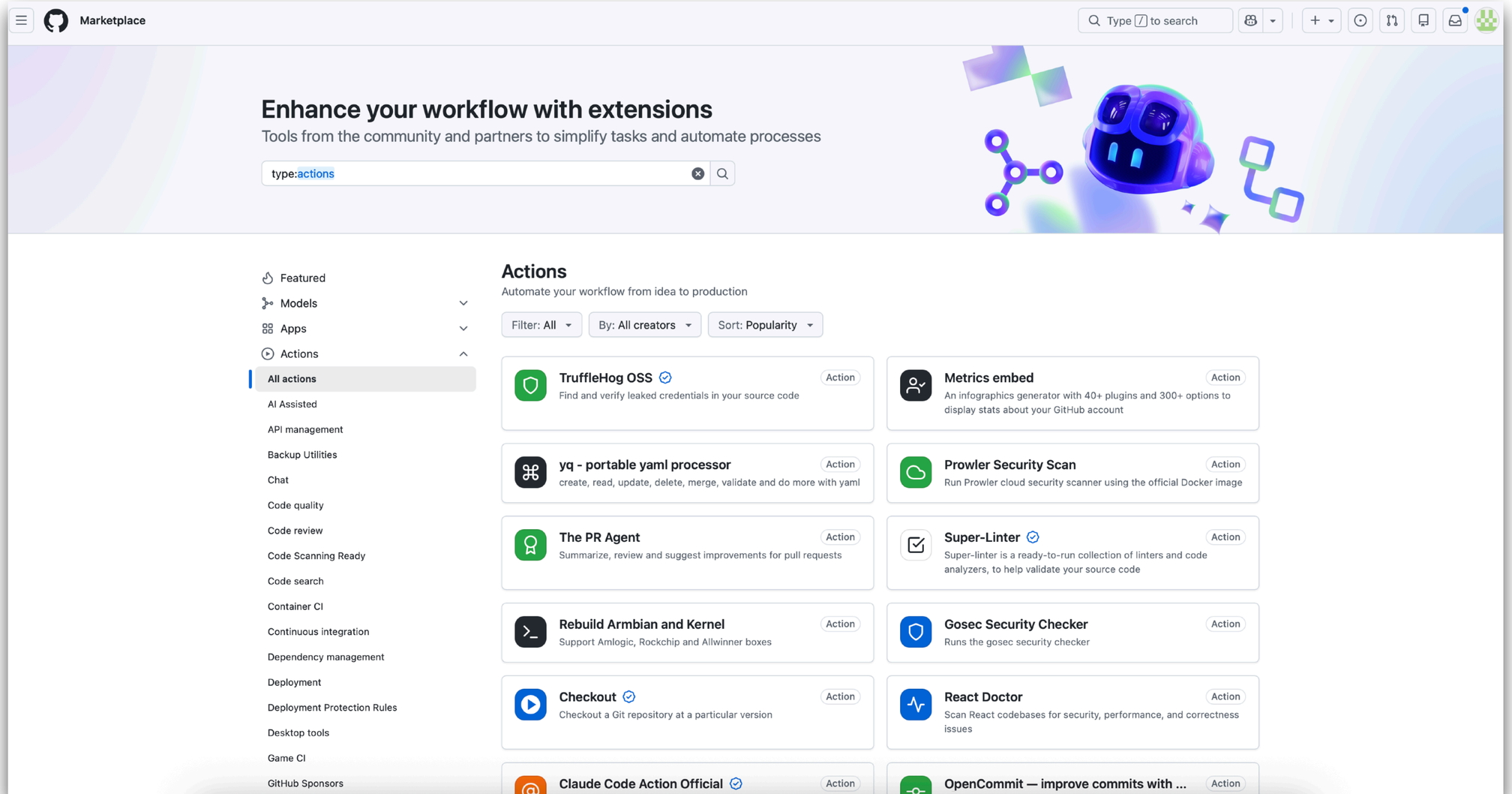
reusable actions (using `uses:`)

“where do I find these?”

GitHub Actions: A Primer

GitHub Marketplace

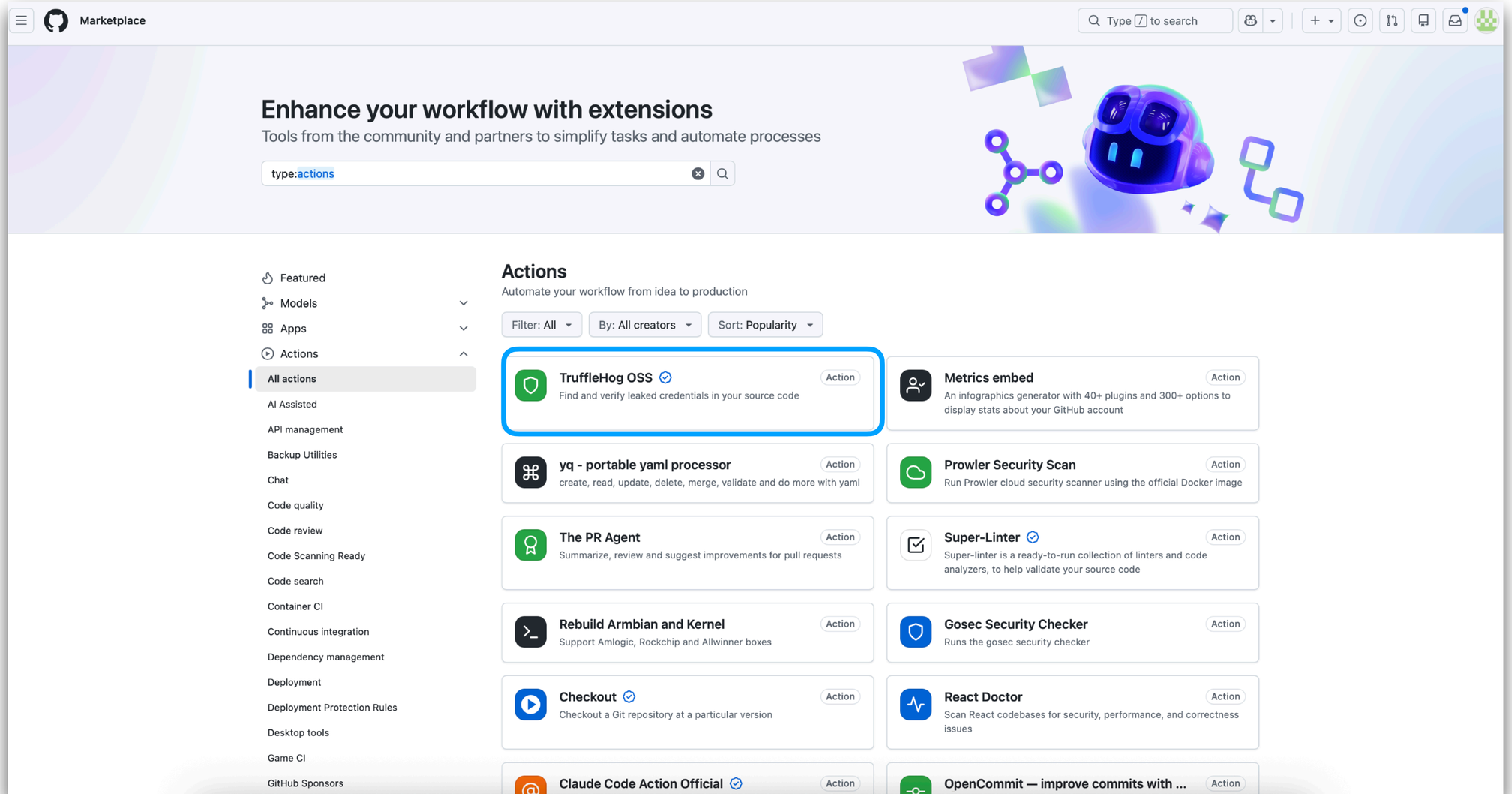
GitHub Actions



GitHub Actions: A Primer

GitHub Marketplace

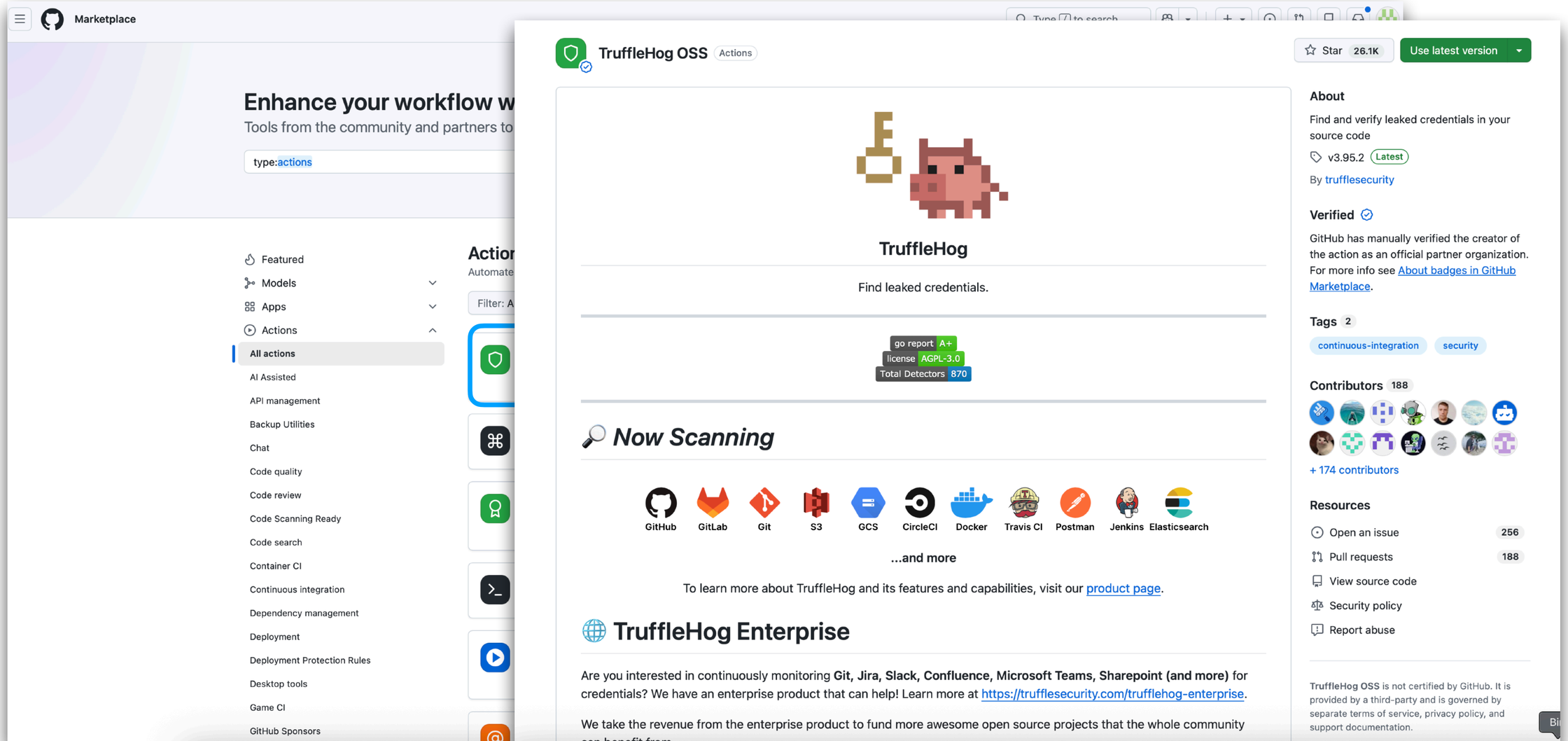
GitHub Actions



GitHub Actions: A Primer

GitHub Marketplace

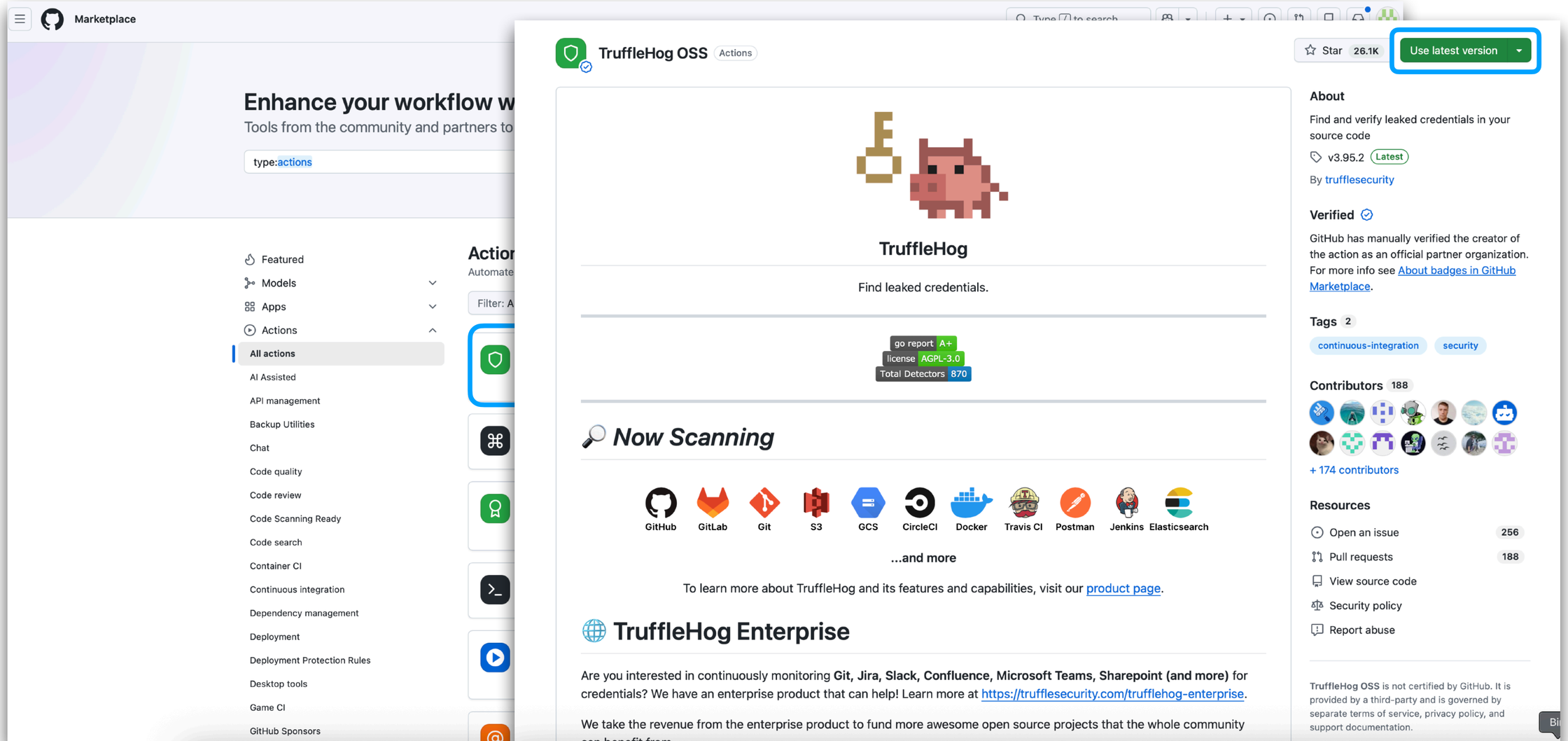
GitHub Actions



GitHub Actions: A Primer

GitHub Marketplace

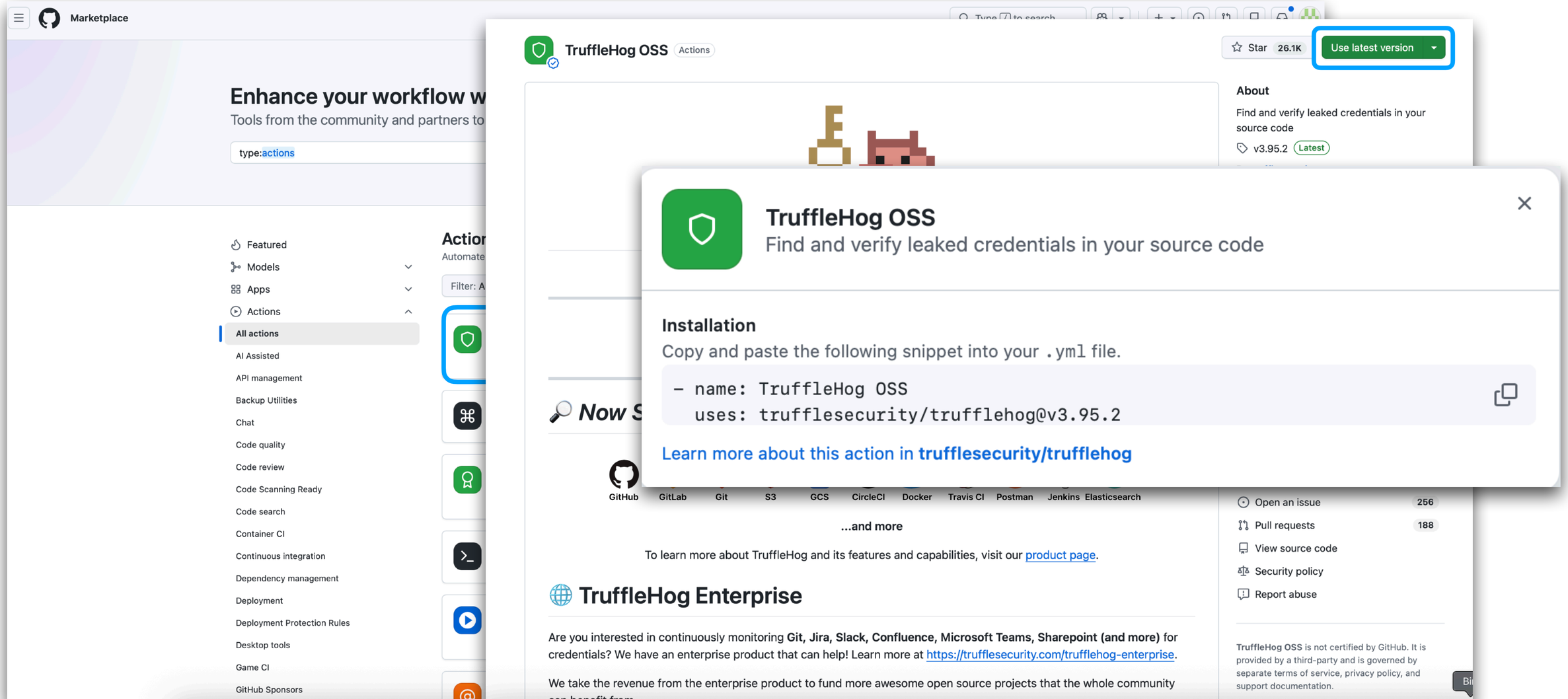
GitHub Actions



GitHub Actions: A Primer

GitHub Marketplace

GitHub Actions



GitHub A…er

Example Wo…

ci.yml

```yaml
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: ...
```

GitHub Actions

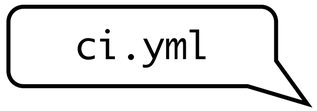
2 kinds of steps …

scripts (using `run:`)

reusable actions (using `uses:`)

GitHub A er

Example Wo




```yaml
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: ...
```

checkout code repository

setup Python 3.11

install platformIO

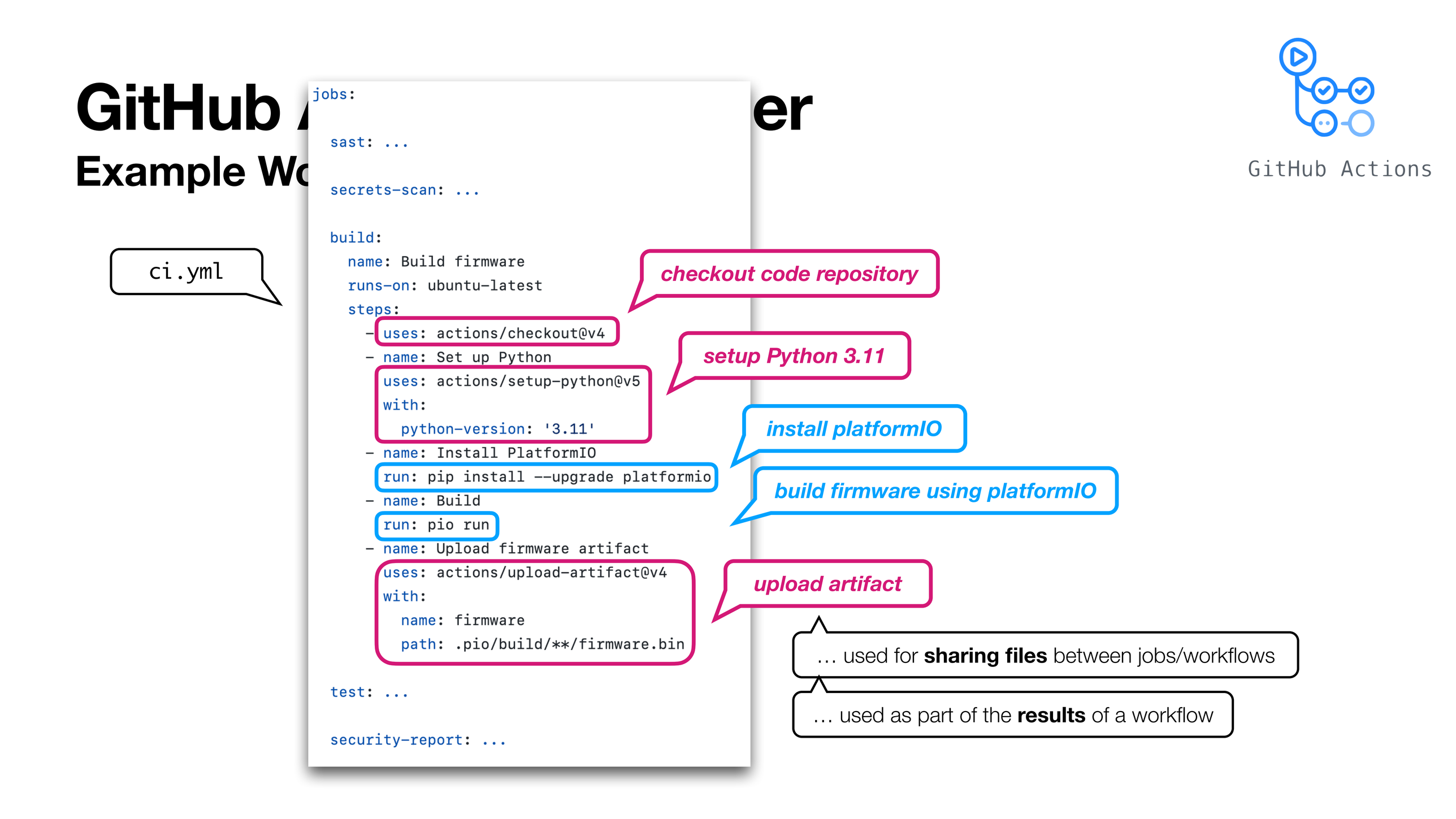
build firmware using platformIO

upload artifact

GitHub Actions

GitHub A…er

Example Wo…



ci.yml

```yaml
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: ...
```

checkout code repository

setup Python 3.11

install platformIO

build firmware using platformIO

upload artifact

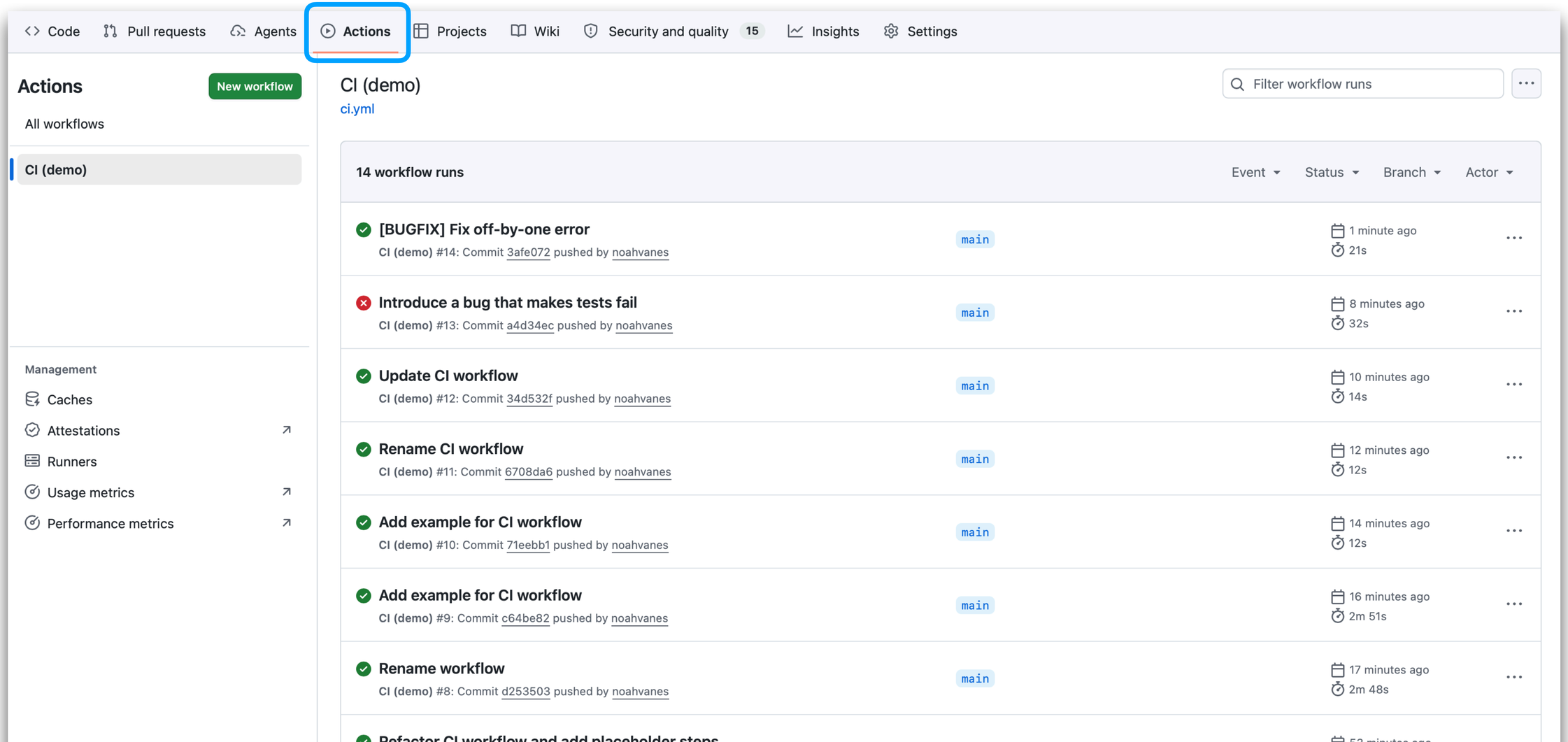
… used for **sharing files** between jobs/workflows

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

GitHub Actions: A Primer

Workflow Results

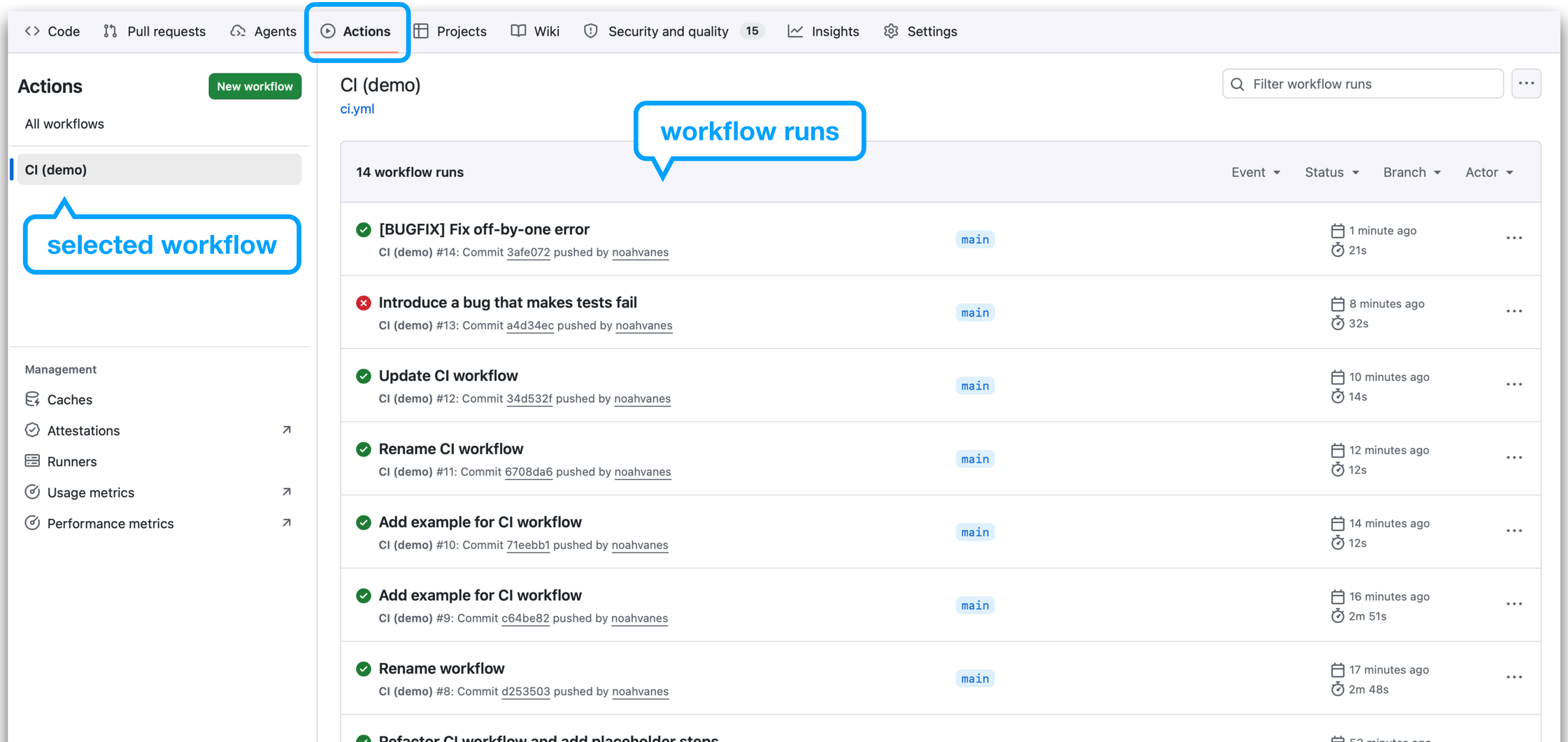
GitHub Actions



GitHub Actions: A Primer

Workflow Results

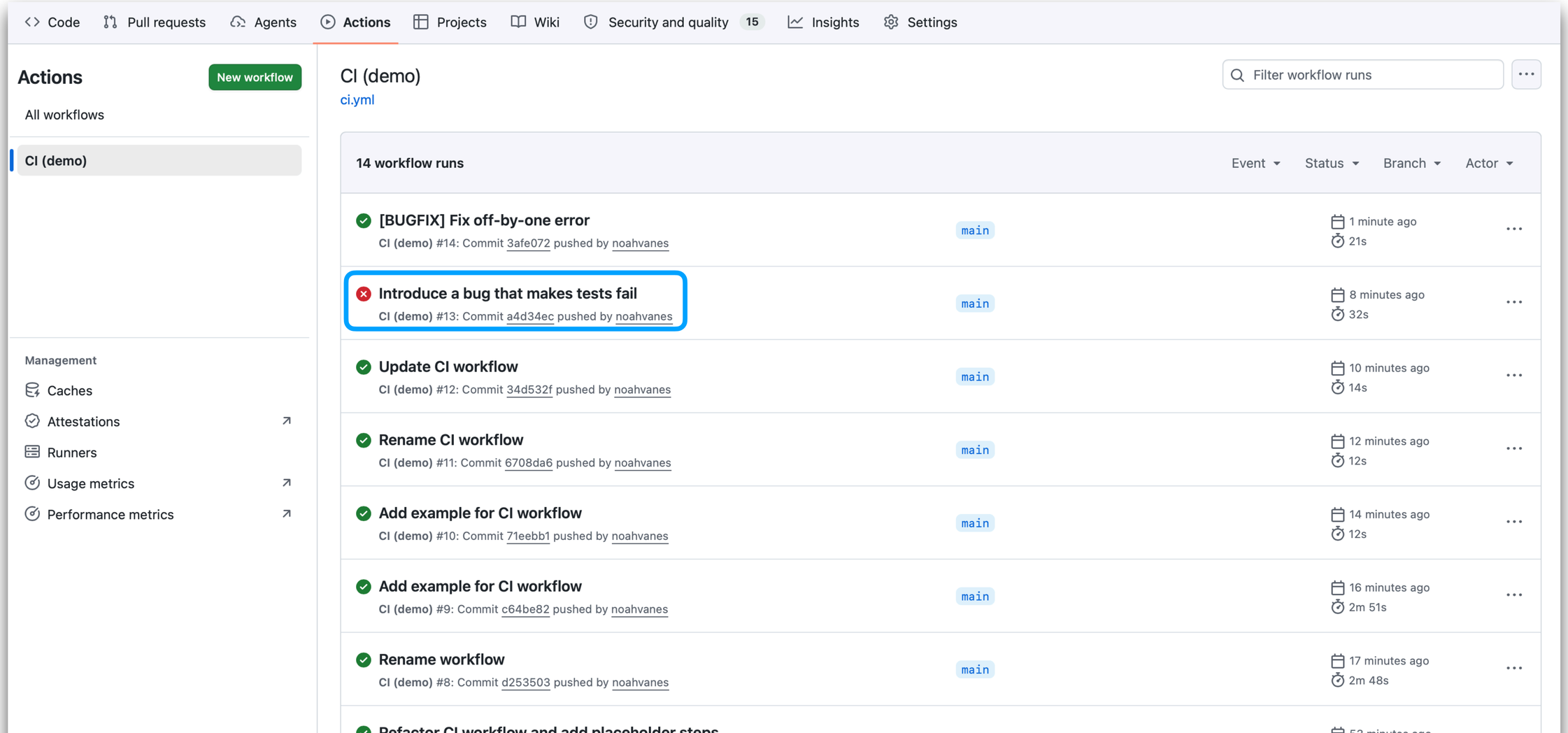
GitHub Actions



GitHub Actions: A Primer

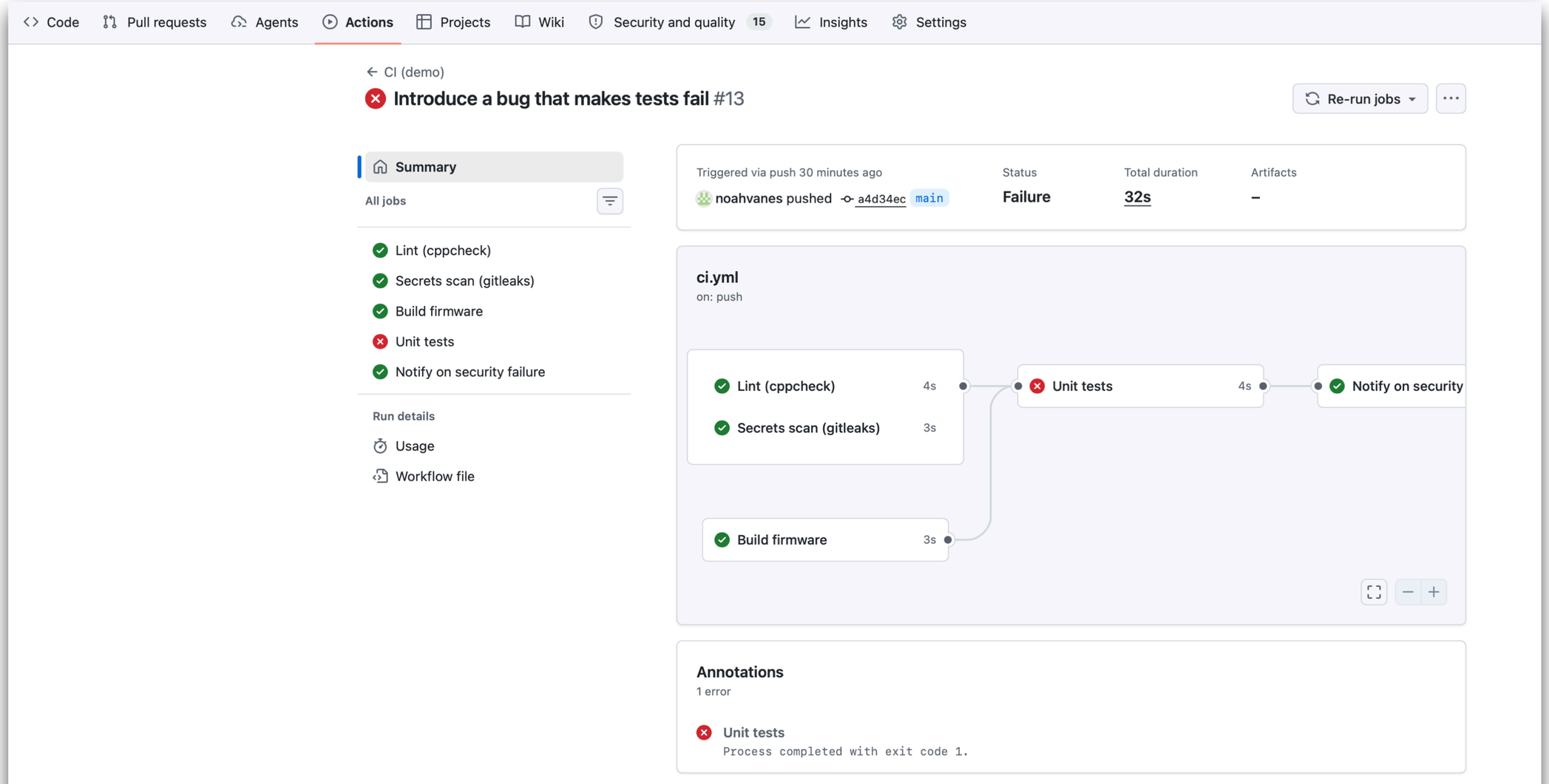
Workflow Results

GitHub Actions



GitHub Actions: A Primer

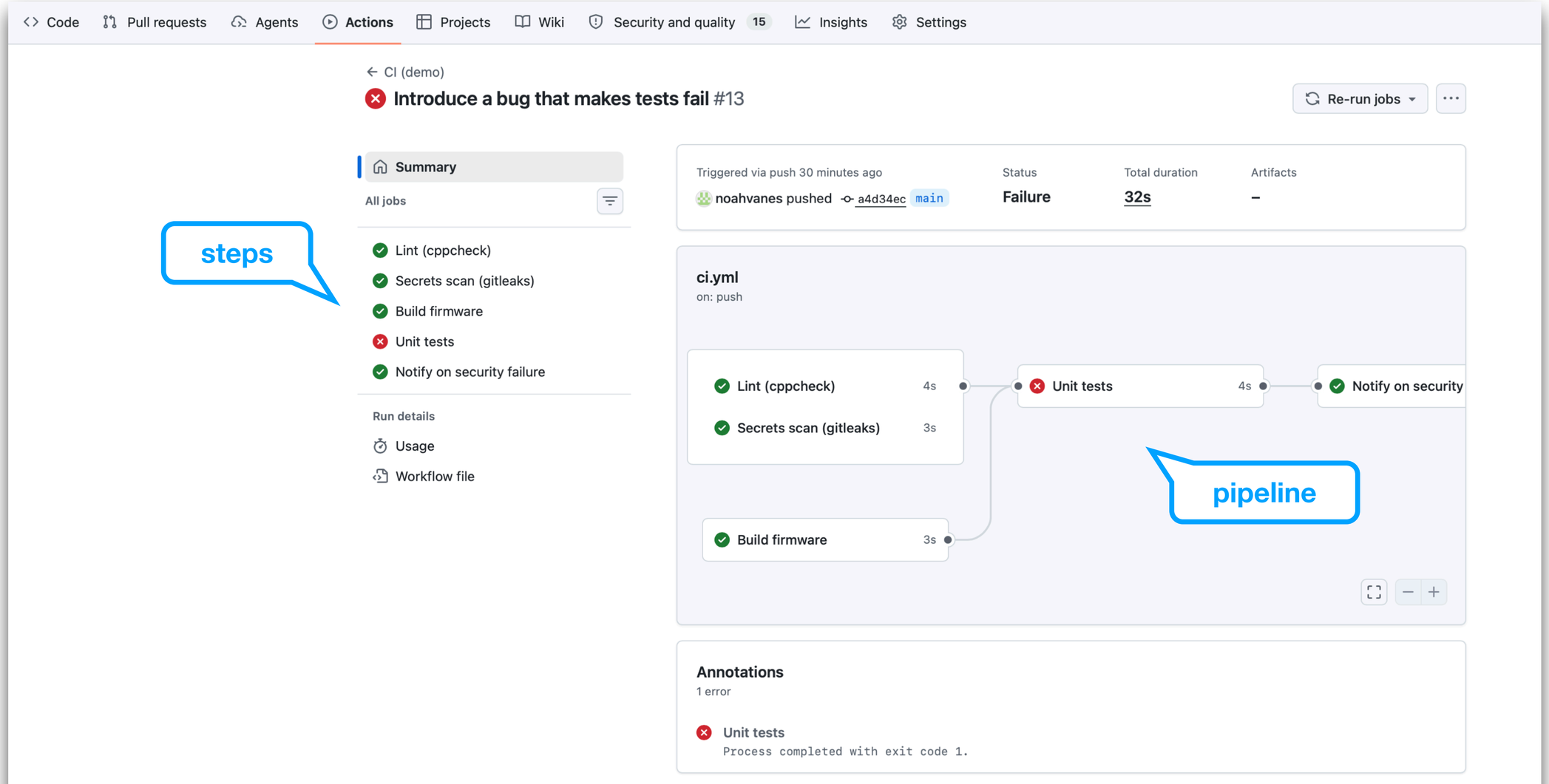
Workflow Results



GitHub Actions: A Primer

Workflow Results

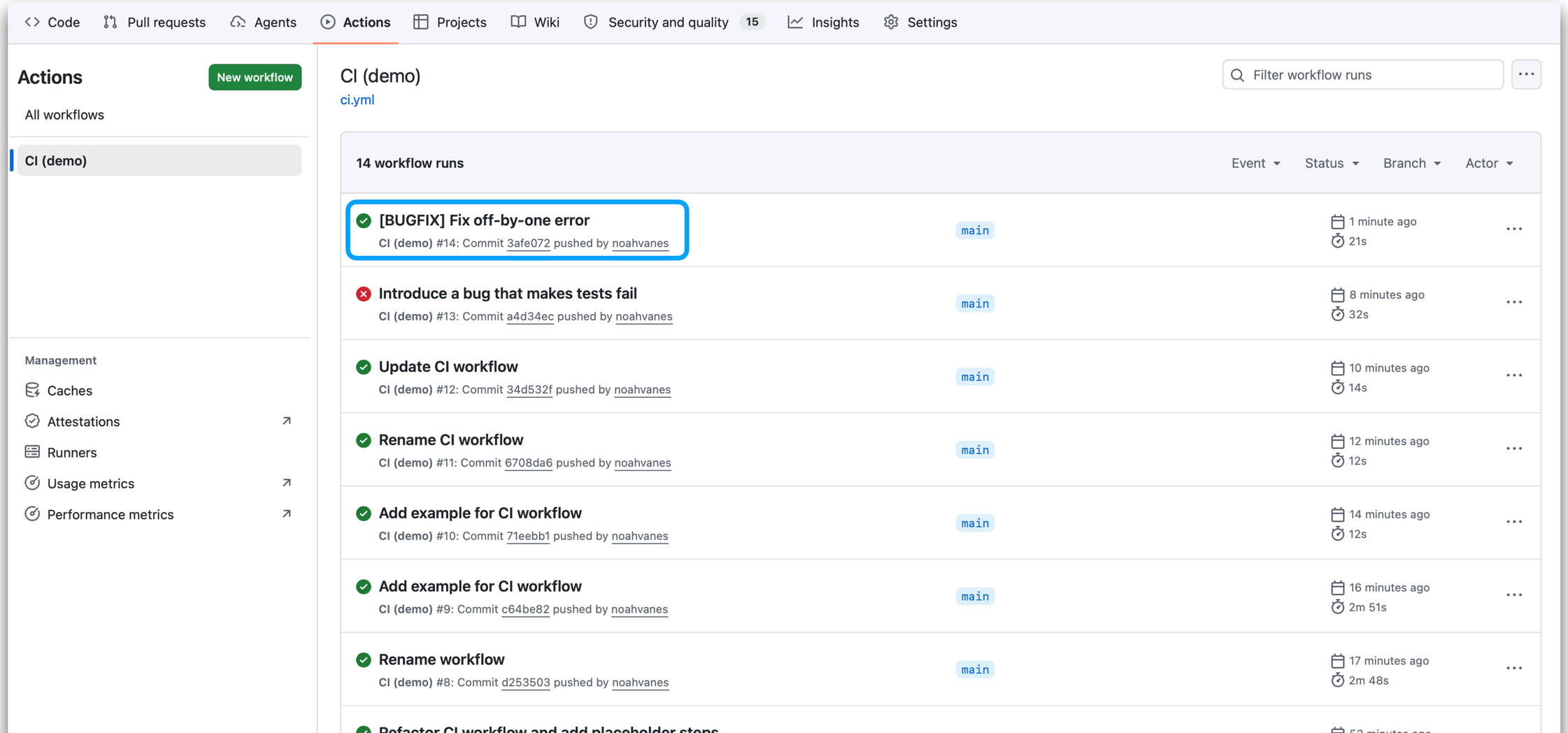
GitHub Actions



GitHub Actions: A Primer

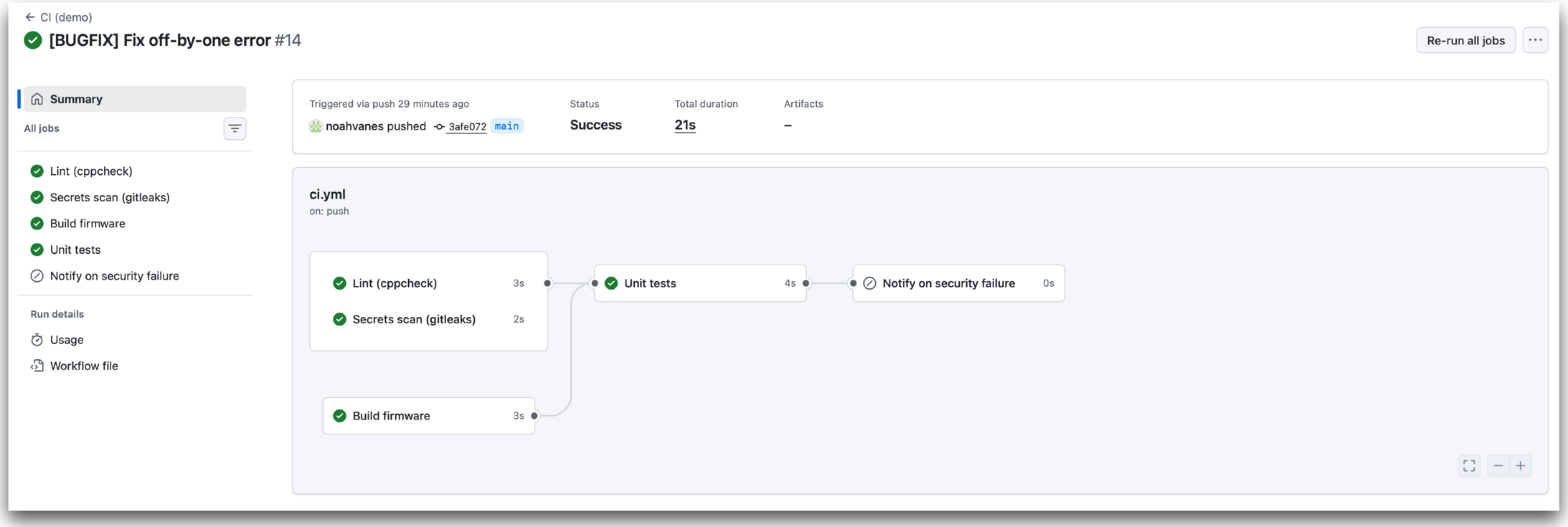
Workflow Results

GitHub Actions



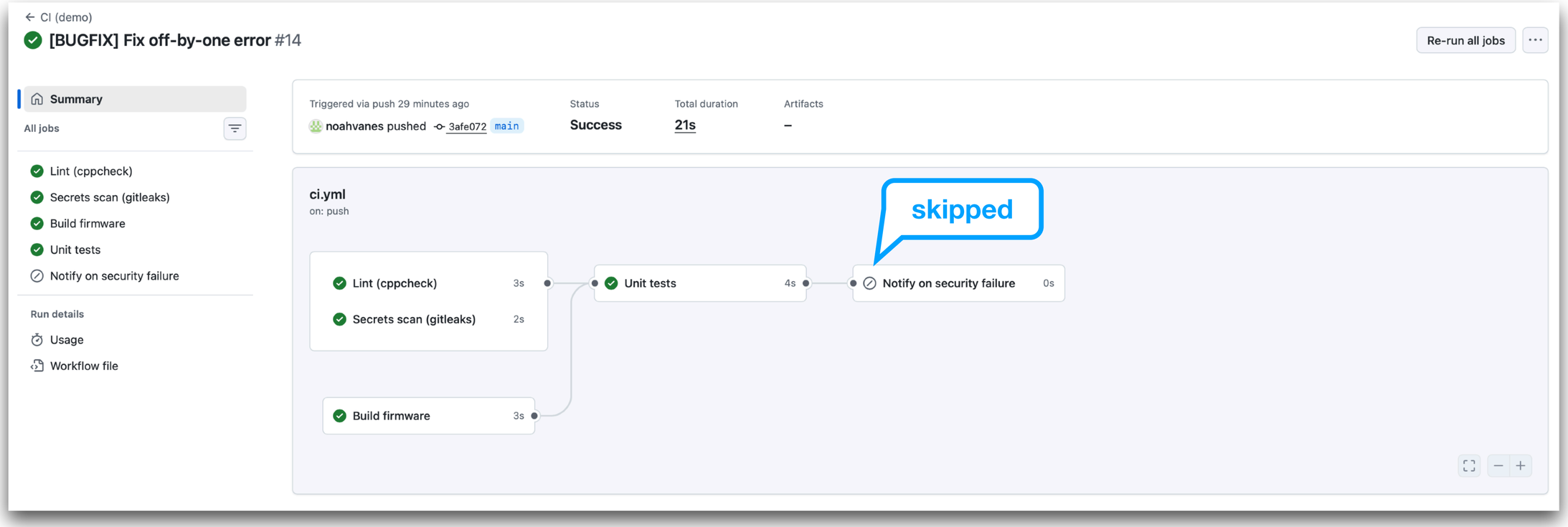
GitHub Actions: A Primer

Workflow Results



GitHub Actions: A Primer

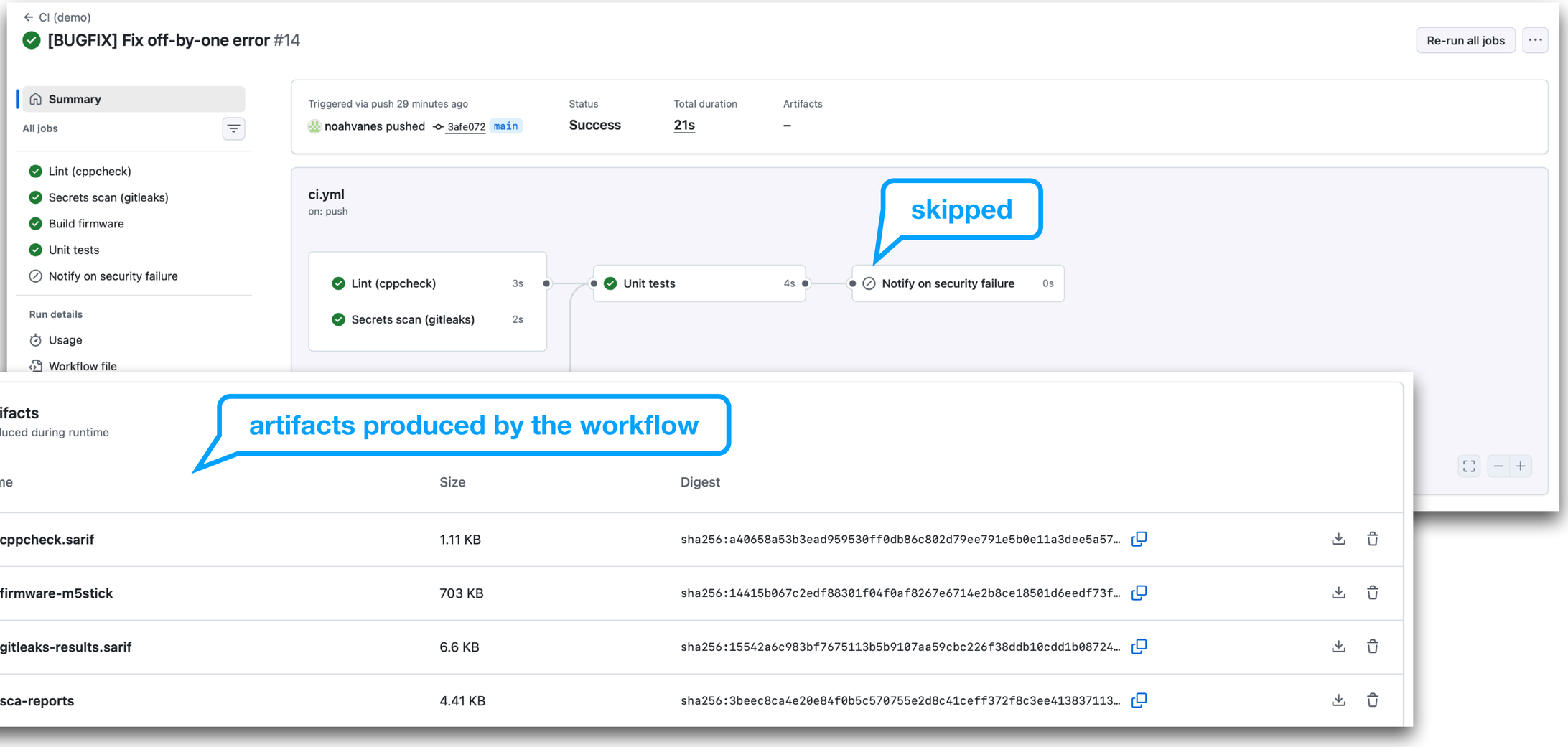
Workflow Results



GitHub Actions: A Primer

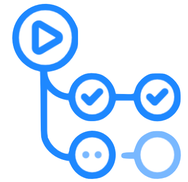
Workflow Results

GitHub Actions



GitHub Actions: A Primer

Secrets Management



GitHub Actions

```yaml
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

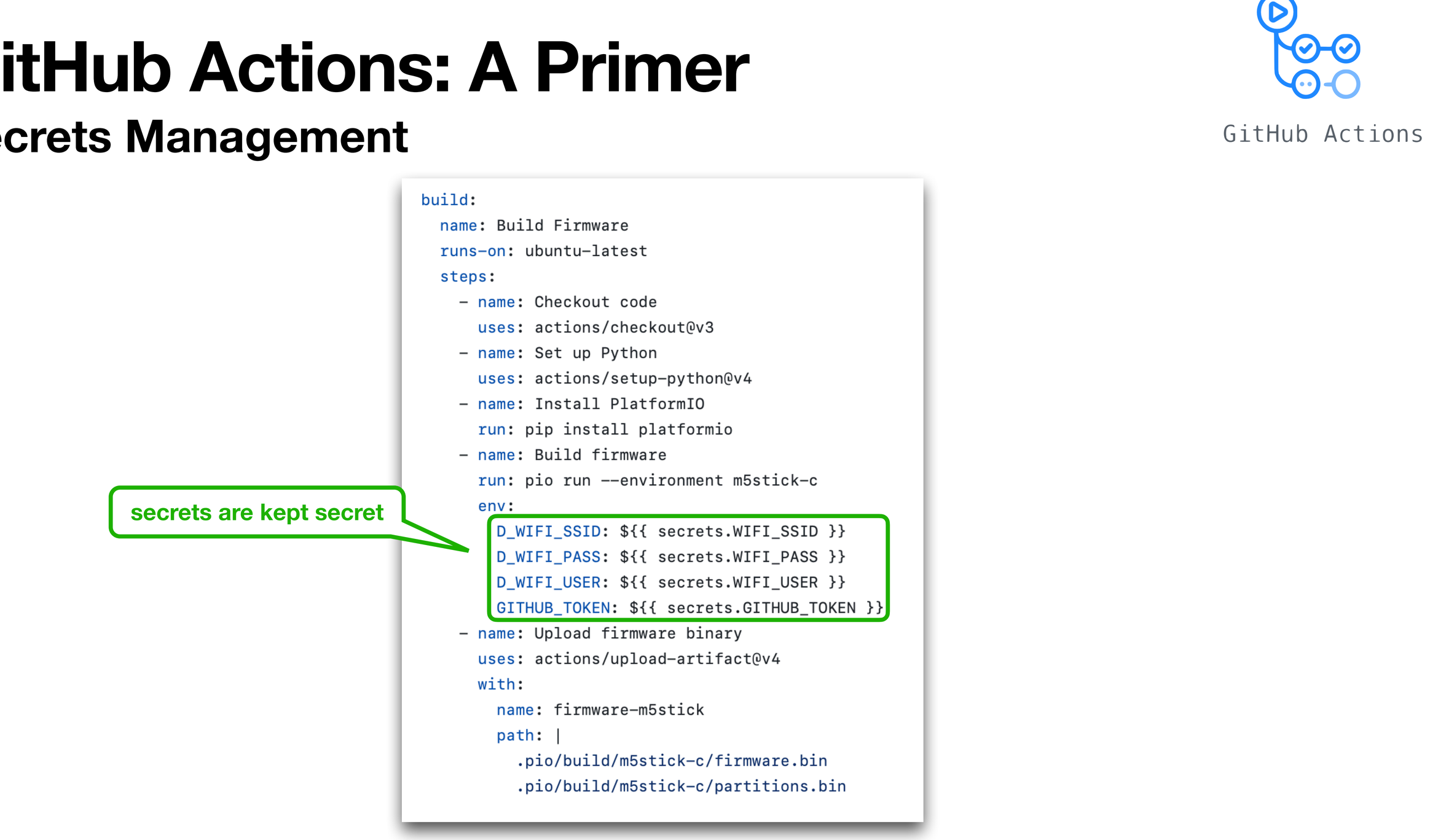
Secrets Management

exposes secrets!

```yaml
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



secrets are kept secret

```yaml
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


```yaml
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


```yaml
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
```

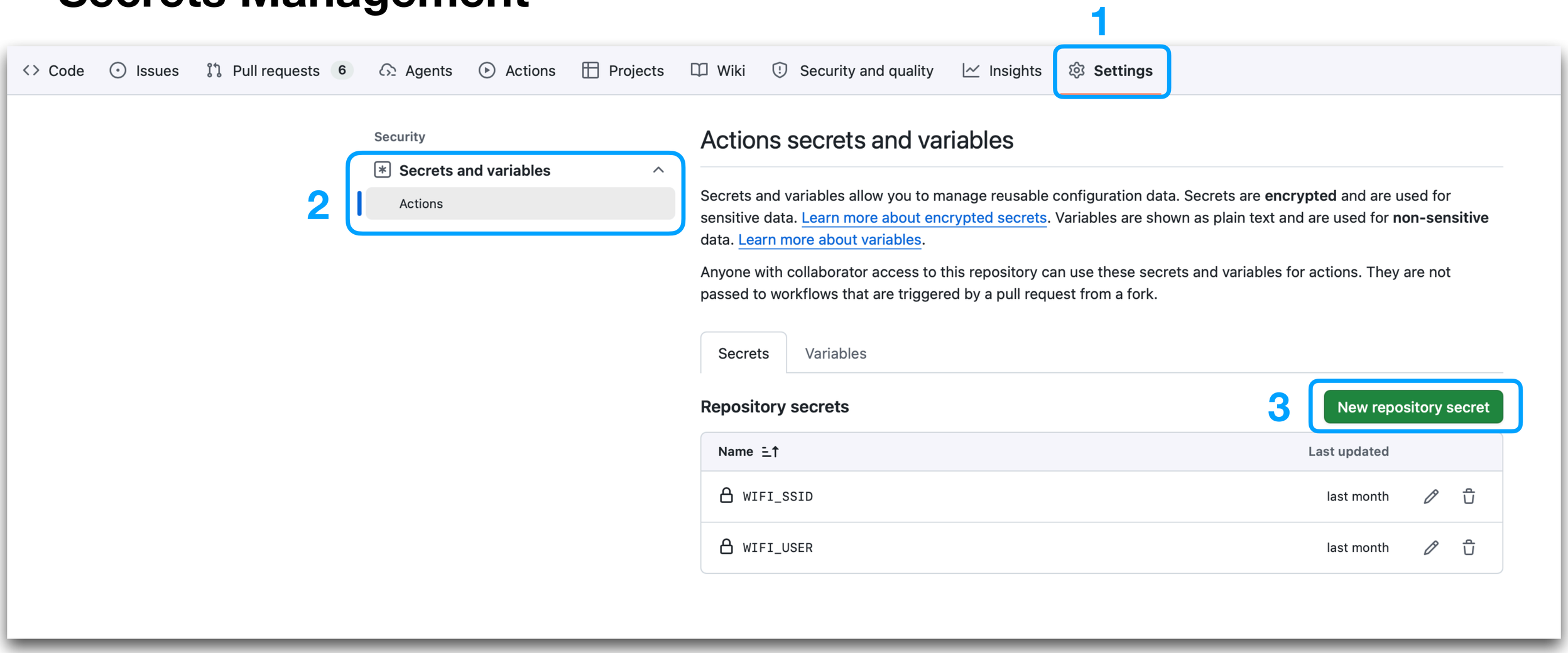
secrets are kept secret

implicit **secrets** object available

Where does this come from?

GitHub Actions: A Primer

Secrets Management



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`

```yaml
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**

GitGuardian

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**

GitGuardian

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


```yaml
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


```yaml
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


```yaml
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

IaC Security Scanning

infrastructure-as-code

HashiCorp
Terraform

IaC Security Scanning

infrastructure-as-code

detect security misconfigurations, compliance violations, …

HashiCorp
Terraform

IaC Security Scanning

infrastructure-as-code

detect security misconfigurations, compliance violations, …

HashiCorp Terraform

SCAnsible

GASEL

tf-check

Terrascan

Snyk

IaC Security Scanning

infrastructure-as-code

detect security misconfigurations, compliance violations, …

HashiCorp Terraform

SCAnsible

GASEL

tf-check

Terrascan

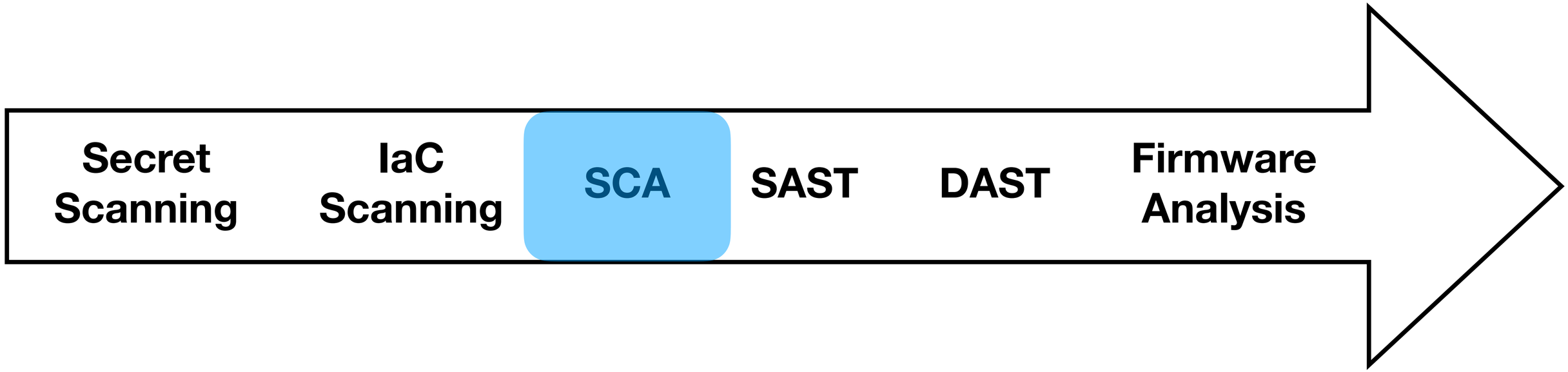
Snyk

Interested? Let us know!

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

A Security-Focused CI/CD Pipeline

Overview



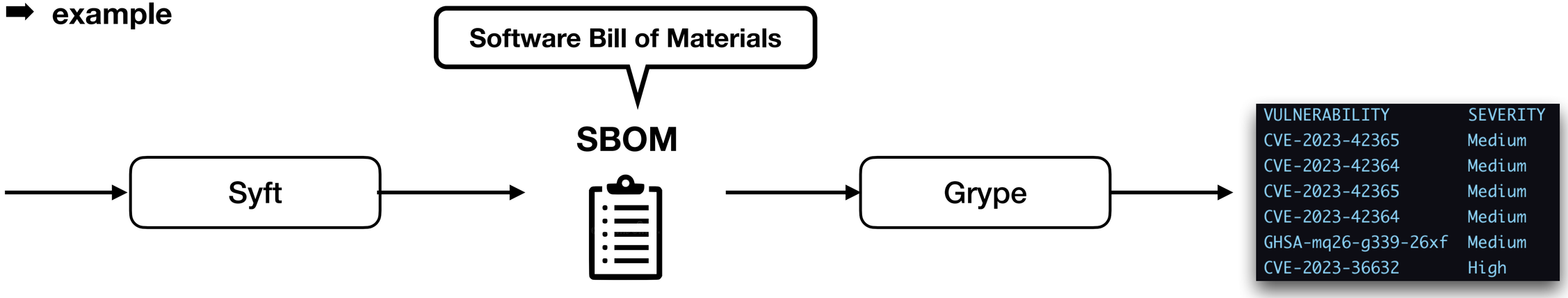
Software Composition Analysis (SCA)

scan dependencies for known vulnerabilities (CVEs)

Software Composition Analysis (SCA)

scan dependencies for known vulnerabilities (CVEs)

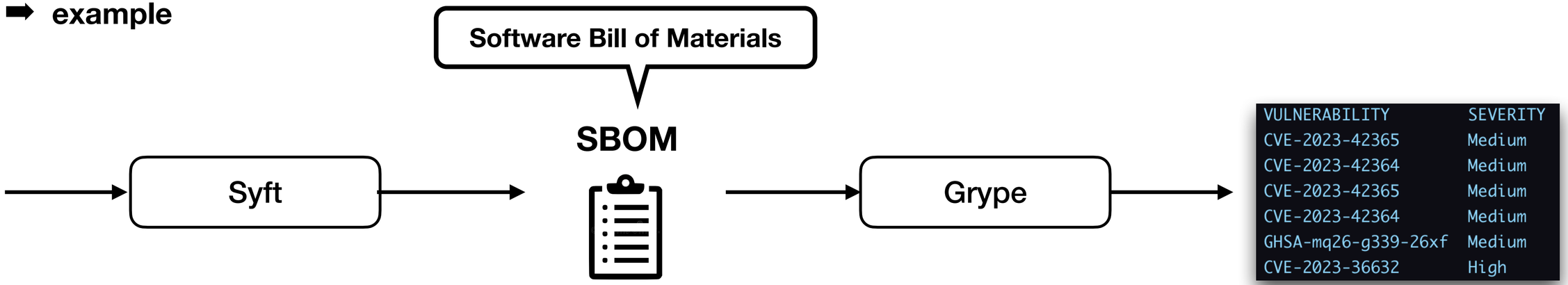
➡ **example**



Software Composition Analysis (SCA)

scan dependencies for known vulnerabilities (CVEs)

➡ **example**




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

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

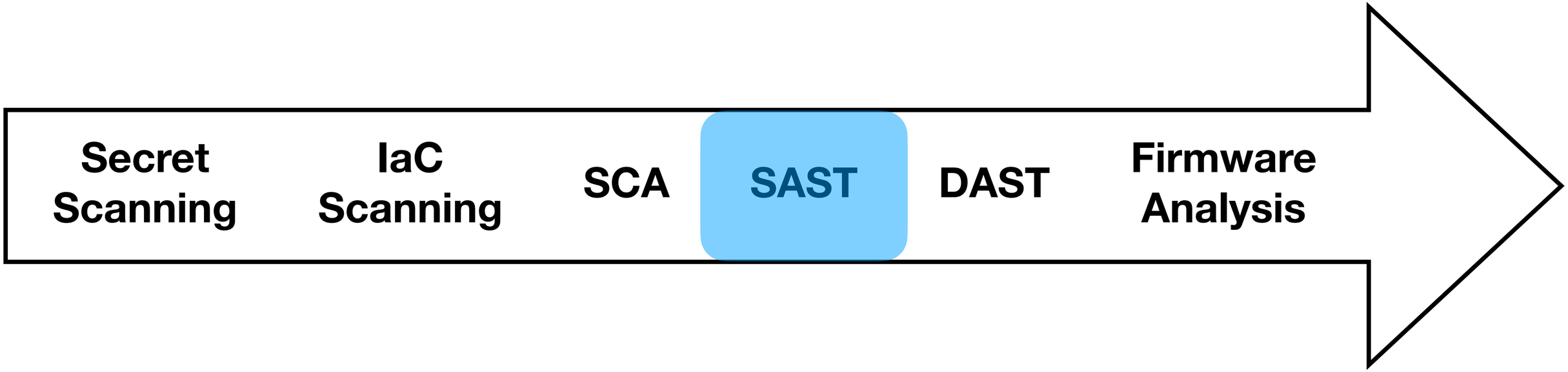
Syft + Grype

```yaml
- 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

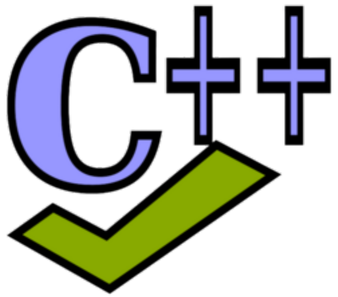
make use of **standard sets of rules / patterns** to look for

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

```yaml
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


```yaml
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


```yaml
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


```yaml
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

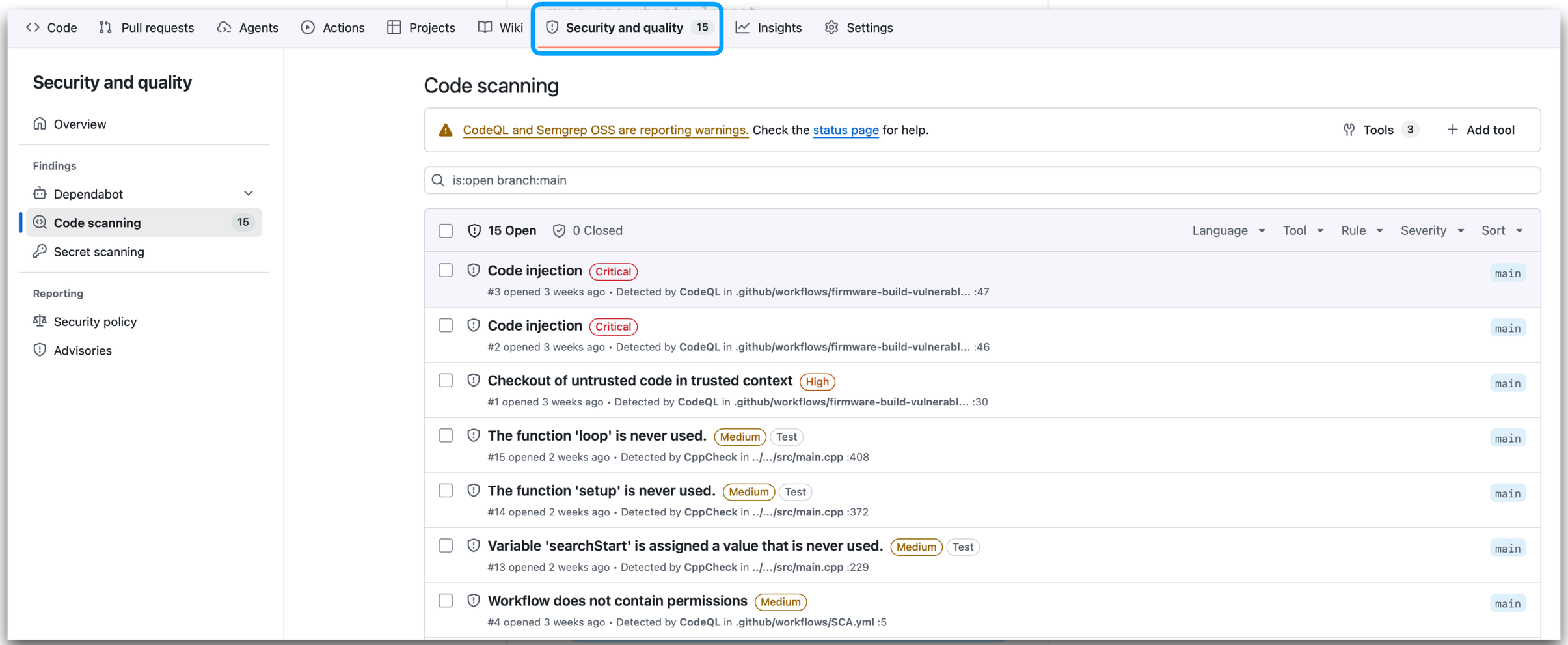
feed SARIF results to GitHub

directly integrated in the GitHub Security tab

Adding SAST to the Pipeline

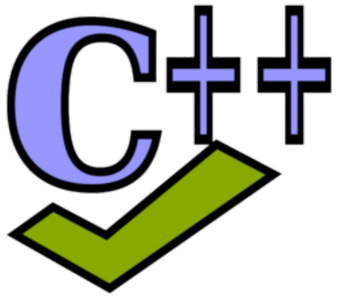
Example: Semgrep

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



Adding SAST to the Pipeline

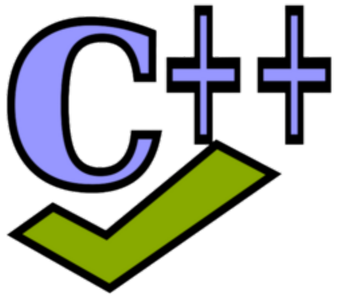
Example: cppcheck



```yaml
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

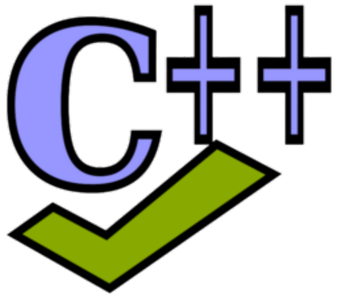
Example: cppcheck



```yaml
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

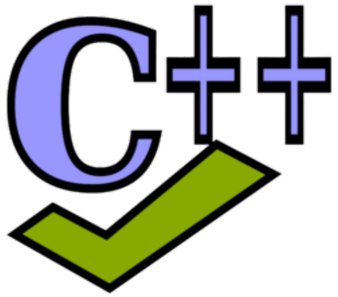


```yaml
sast-cppcheck:
  name: SAST (cppcheck)
  runs-on: ubuntu-latest
  steps:
    - name: Checkout code
      uses: actions/checkout@v4
    - name: Restore cppcheck cache
      id: cache
      uses: actions/cache@v4
      with:
        path: ~/.local
        key: cppcheck-${{ runner.os }}-v1
    - name: Build cppcheck from source
      if: steps.cache.outputs.cache-hit != 'true'
      run: |
        sudo apt update && sudo apt install -y build-essential cmake libpcre3-dev libtinyxml2-dev git
        git clone https://github.com/cppcheck-opensource/cppcheck.git
        cd cppcheck
        mkdir build && cd build
        cmake -DCMAKE_INSTALL_PREFIX=$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



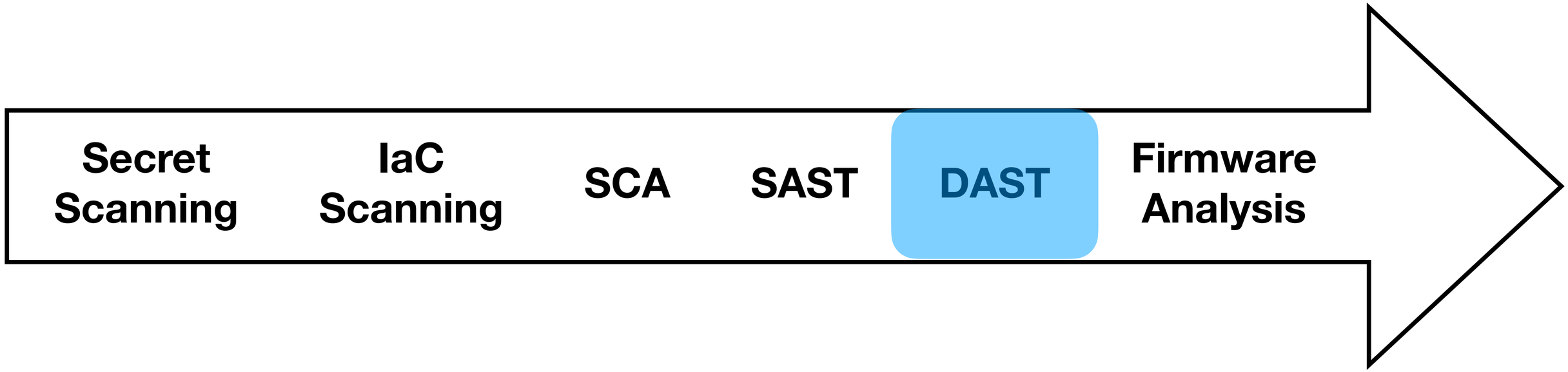

```yaml
sast-cppcheck:
  name: SAST (cppcheck)
  runs-on: ubuntu-latest
  steps:
    - name: Checkout code
      uses: actions/checkout@v4
    - name: Restore cppcheck cache
      id: cache
      uses: actions/cache@v4
      with:
        path: ~/.local
        key: cppcheck-${{ runner.os }}-v1
    - name: Build cppcheck from source
      if: steps.cache.outputs.cache-hit != 'true'
      run: |
        sudo apt update && sudo apt install -y build-essential cmake libpcre3-dev libtinyxml2-dev git
        git clone https://github.com/cppcheck-opensource/cppcheck.git
        cd cppcheck
        mkdir build && cd build
        cmake -DCMAKE_INSTALL_PREFIX=$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**

boofuzz

➡ **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**

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.

EMBA

Firmware Analysis

source-code level and **binary analysis**

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

EMBA

Interested? Let us know for future workshops!

A Security-Focused CI/CD Pipeline

Overview

A Security-Focused CI/CD Pipeline

Overview