GitHub Actions

Matt Watkins
Release Engineering
The Linux Foundation

https://lfnetworking.org
Personal Introduction

- BSc Degree, Physics with Astrophysics, York University
- Graduated in 1997 and moved to Cambridge
- Accidentally became an Internet network engineer
- Lots of data centre work; routers, switches and firewalls
- Worked for a number of Internet Service Providers
- Sun Solaris, Linux system administration
- 25 Years in networking business, recent pivot into software
- Two years working for the Linux Foundation
• Brief Review of Existing LFN/Project Tooling (Gerrit, Jenkins, JCasC, JJB, Global JJB, Sandbox Access)
• GitHub and GitHub Actions Overview
• Digging into GHA Details (Workflow Files, Linting/Verification, Triggers, Variables/Secrets)
• Advanced Features
  I/O, Artefacts, Signing, Trusted Publishers, Matrix Operations, Apps
• Composite Actions, Reusable Workflows
• Interactive Demonstrations throughout!
DevOps Statement of Intent

“The mindset we should carry is that we always want to automate ourselves into a better job. We want to make sure that the task we’re doing manually today becomes mostly automated”
Review Existing Tooling

**Gerrit <-> GitHub**

- GIT backend with web portal/interface
- Considered best in class for code review purposes
- Not going anywhere anytime soon for LFN projects
- In most cases, code is already replicated to GitHub
- Integration was held back for some time by missing APIs/features
- Last round of Gerrit updates unlocked integration capability
- This means there is an opportunity to deploy GitHub Actions
- However, if replication is performing poorly, problems can arise
Review Existing Tooling

Jenkins, JCasC, JJB, Global JJB

- Jenkins hosted in VEXXHOST and jobs run on an isolated network
- Executor nodes exist in a pool; if not available spin-up is slow
- GitHub Actions uses containers and jobs deploy/execute quickly
- Potentially easier to tap pre-built images, e.g. Docker Hub, etc.
- Puppet is used for server management
- Jenkins configuration not defined using interface; JJB and JCasC
- Job templates in global-jjb (sub-repo), ci-management contains jobs
Features:

- Configuration files authored in YAML format
- Located in the directory: .github/workflows
- Sit alongside the repository code
- Can pull/execute actions from other repositories
- YAML and GHA linting tools are available
GitHub Actions Workflows

Triggers

Documentation: Events that trigger workflows

• Can be triggered manually
• Can be triggered automatically on a variety of events
• Can be run on a schedule (CRON)
Workflow Triggers

Useful Examples

Manual
workflow_dispatch:

Schedule
schedule:
  - cron: "0 0 * * MON"

Repository actions
pull_request:
  types: [opened, reopened, edited, synchronize]

Pushing tags
push:
  # Only invoked on release tag pushes
tags:
  - v*.*.*
Example Workflow Configuration

jobs:
  build:
    name: "Audit Python dependencies"
    runs-on: ubuntu-latest
    strategy:
      fail-fast: false
      matrix:
        python-version: ["3.9", "3.10", "3.11"]
Release Job for Python Project

https://github.com/os-climate/ITR-examples/actions/runs/6787713863
Development Release Job Demo

https://github.com/os-climate/ITR-examples/actions/workflows/test-release.yaml
Trusted Publishers

Allows interactions between sites without use of fixed authentication tokens

Links:
- OpenID
- PyPI
- GitHub

Trusted Publisher Management

OpenID Connect (OIDC) provides a flexible, credential-free mechanism for delegating publishing authority for a PyPI package to a trusted third party service, like GitHub Actions.

PyPI users and projects can use trusted publishers to automate their release processes, without needing to use API tokens or passwords.

You can read more about trusted publishers and how to use them [here](#).

Manage current publishers

<table>
<thead>
<tr>
<th>Publisher</th>
<th>Repository</th>
<th>Workflow</th>
<th>Environment name</th>
</tr>
</thead>
<tbody>
<tr>
<td>GitHub</td>
<td>os-climate/ITR</td>
<td>release.yaml</td>
<td>pypi</td>
</tr>
</tbody>
</table>
Example Implementation in workflow:

```yaml
jobs:
pypi-publish:
  name: upload release to PyPI
  runs-on: ubuntu-latest
+  permissions:
+    # IMPORTANT: this permission is mandatory for trusted publishing
+    id-token: write
steps:
  # retrieve your distributions here

  - name: Publish package distributions to PyPI
    uses: pypa/gh-action-pypi-publish@release/v1
    with:
      username: __token__
      password: ${{ secrets.PYPI_TOKEN }}
```
# Installed GitHub Apps

GitHub Apps augment and extend your workflows on GitHub with commercial, open source, and homegrown tools.

<table>
<thead>
<tr>
<th>App</th>
<th>Developed by</th>
<th>Configure</th>
</tr>
</thead>
<tbody>
<tr>
<td>DCO</td>
<td>dcoapp</td>
<td></td>
</tr>
<tr>
<td>LFx Security GitHub App</td>
<td>LF-Engineering</td>
<td></td>
</tr>
<tr>
<td>pre-commit ci</td>
<td>pre-commit-ci</td>
<td></td>
</tr>
<tr>
<td>Slack</td>
<td>github</td>
<td></td>
</tr>
</tbody>
</table>
Can work with pre-commit hooks!

Configure application:
https://github.com/organizations/os-climate/settings/installations/43027599

Take a look at a typical pull request:
https://github.com/os-climate/ITR/pull/298

Pre-commit output:
https://results.pre-commit.ci/run/github/384066937/1699962571.5gI3qZy5Q7aBzBj9J-fmgw

Can auto-update linting tools by raising a PR:
https://github.com/os-climate/ITR/pull/274
Documentation on configuration:
Reusable Workflows

What Are they?

Reusable workflows are YAML-formatted files, very similar to any other GitHub Actions workflow files. As with other workflows, you locate reusable workflows in the “.github/workflows” directory inside a repository. Subdirectories of this folder are NOT supported.

For a workflow to be reusable, the values for on must include “workflow_call”:

```yaml
on:
  workflow_call:
```
Why should you use them?

- **Reduces redundancy**
  
  If you have multiple repositories deployed the same way, reusable workflows can help you keep them in sync.

- **No Duplication, modularity**
  
  We already know that by referencing workflows in another GitHub Action workflow, you reproduce the same work.

- **Easy to create**
  
  All you need to have is a trigger and a workflow_call to prompt it. This simple and effortless process is documented here.
Reusable Workflows

Features of Reusable Workflows

Three primary features:

1. Inputs: variables/other data passed in by the calling workflow
2. Secrets: credentials that can be consumed by the workflow
3. Outputs: artefacts or other data created by the workflow

Inputs/secrets can be mandatory requirements!

E.g. required: true
Reusable Workflows

release Engineering: Reusable Workflows

We have a new repository, it’s public, and you can find it here:

Release Engineering are porting our jobs!

You may have already seen them in Gerrit!

e.g.
https://git.opendaylight.org/gerrit/c/releng/builder/+/108980

...and here:
https://github.com/opendaylight/releng-builder/actions/runs/6878770145
Run Workflows Locally

Useful for debugging; it can be done!

The tools is here: https://github.com/nektos/act

Prerequisites:
• A local Docker install
• A suitable base image to execute the workflows with

Obviously, there are some limitations…
Mainly, the lack of full GitHub environment/context
e.g. Missing secrets/tokens, trusted publishing support
Running Actions Locally

User documentation: [https://nektosact.com/](https://nektosact.com/)

Configuration file specifies the container image: ~/.actrc

e.g.

```
-P ubuntu-latest=catthehacker/ubuntu:full-latest
```

Have encountered some issues with Apple Silicon…
Docker Image Requirements

- Choose same baseline OS image(s) as your workflows
- Modern NodeJS required
- Python3 and related tools (pyenv/pip/venv)

Apple Silicon issues: some third party workflows download x64 binaries and ignore the underlying platform!
Run Workflows Locally

Example:

```
INFO[0000] Parallel tasks (0) below minimum, setting to 1
  Production build and release/Build packages
  Production build and release/Build packages

INFO[0000] Parallel tasks (0) below minimum, setting to 1
  Production build and release/Build packages
  Production build and release/Build packages

INFO[0000] Parallel tasks (0) below minimum, setting to 1
  Production build and release/Build packages
  Production build and release/Build packages

INFO[0000] Parallel tasks (0) below minimum, setting to 1
  Production build and release/Build packages
  Production build and release/Build packages
```

```bash
# Start image=mattwatkins1/ubuntu2204x64:20231114
```

```bash
# docker pull image=mattwatkins1/ubuntu2204x64:20231114 platform=linux/arm64 username=forcePull=false
```

```
# docker run image=mattwatkins1/ubuntu2204x64:20231114 platform=linux/arm64 entrypoint="["tail" -f "*/dev/null"] cmd=[] network="host"
```

```
# git clone https://github.com/pdm-project/setup-pdm' # ref=v3
# git clone https://github.com/actions/setup-python' # ref=v4.7.0
# Run Pre Sign packages with Sigstore
  - git clone https://github.com/actions/upload-artifact' # ref=v3
  - git clone https://github.com/softprops/action-gh-release' # ref=v1
  - Success - Pre Sign packages with Sigstore
```

```
```

```
# Success - Main Checkout repository
```

```
# Run Main Setup RPM for build commands
```

Further Reading

Want to learn more?

Bookmarks for your GitHub Actions learning journey 🚀
Check out these protips from @talktopri
Get started learning GitHub Actions in 3 easy steps:
• More details on CI/CD 🔒
• Explore the GitHub Actions Documentation
• GitHub Actions public roadmap
Thank You!