

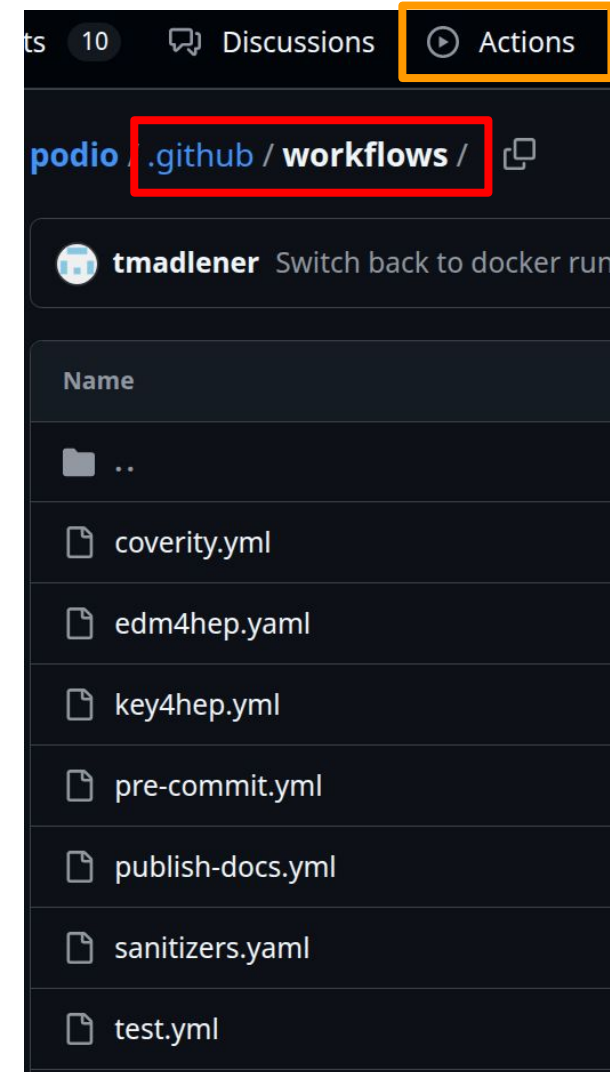
(Ab)using github actions for fun and profit

FH SciComp Workshop 2025

Federico Meloni & Thomas Madlener
July 3, 2025

github actions basics

- [github actions](#) is githubs automation platform for enabling CI/CD workflows
- Similar to [GitLab CI/CD](#)
- YAML files in `.github/workflows` are considered to be github action workflows
 - Run automatically(!) by default
- Fully integrated into the github ecosystem
- Free runners available for public and private repositories



github actions basic

The basic anatomy of a workflow

- Defines what to run when and where
- Workflow name & job names are “free form”
- Flexible trigger system
- Job matrices are possible

```
name: Key4hep build

on:
  push:
    branches:
      - main
  pull_request:
  workflow_dispatch:
  schedule:
    - cron: '16 4 * * *'

jobs:
  build:
    strategy:
      matrix:
        build_type: ["release", "nightly"]
        image: ["alma9"]
        stack: ["key4hep"]
        include:
          - build_type: nightly
            image: ubuntu24
            stack: key4hep
    fail-fast: false
    runs-on: ubuntu-latest

    steps:
      - uses: actions/checkout@v4
      - uses: key4hep/key4hep-actions/cache-external-data@main
      - uses: key4hep/key4hep-actions/key4hep-build@main
      with:
        build_type: ${{ matrix.build_type }}
        image: ${{ matrix.image }}
        stack: ${{ matrix.stack }}
```

Triggers

Job Matrix

Job steps

build (nightly, alma9, key4hep)
succeeded 2 hours ago in 5m 52s

| Step | Duration |
|---|----------|
| Set up job | 2s |
| Run actions/checkout@v4 | 1s |
| Run key4hep/key4hep-actions/cache-external-data@main | 0s |
| Run key4hep/key4hep-actions/key4hep-build@main | 5m 44s |
| Post Run key4hep/key4hep-actions/key4hep-build@main | 3s |
| Post Run key4hep/key4hep-actions/cache-external-data@main | 0s |
| Post Run actions/checkout@v4 | 1s |
| Complete job | 0s |

github actions basics

Defining steps to run

- A step can run pretty much anything
 - Shell command, script, executable, ...
- Each step starts with the original environment
 - Changes to file system persist
- Can also use a **reusable action**
- Each gets a separate (collapsible) log entry

```
jobs:
  build:
    runs-on: ubuntu-latest
    strategy:
      fail-fast: false
    steps:
      - uses: actions/checkout@v4
      - uses: actions/setup-python@v5
        with:
          python-version: 3.12
      - name: Install Requirements
        run: |
          pip install --upgrade pip
          pip install -r requirements.txt
      - name: Fetch external sources
        run: |
          bash .github/scripts/fetch_external_sources.sh
      - name: Sphinx build
        run: |
          sphinx-build -M html docs build
          sphinx-build -b linkcheck docs linkcheck
      - uses: actions/upload-pages-artifact@v3
        with:
          path: build/html
```

- > ✓ Set up job
- > ✓ Run actions/checkout@v4
- > ✓ Run actions/setup-python@v5
- > ✓ Install Requirements
- > ✓ Fetch external sources
- > ✓ Sphinx build
- > ✓ Run actions/upload-pages-artifact@v3
- > ✓ Post Run actions/setup-python@v5
- > ✓ Post Run actions/checkout@v4
- > ✓ Complete job

✓ Install Requirements

```
1 ▶ Run pip install --upgrade pip
12 Requirement already satisfied: pip in /opt/hostedtoolcache/Python312/3.12.0/x64/lib/python3.12/site-packages (24.0)
13 Collecting alabaster==0.7.13 (from -r requirements.txt)
14   Downloading alabaster-0.7.13-py3-none-any.whl.metadata (2.5 kB)
15 Collecting attrs==23.1.0 (from -r requirements.txt)
16   Downloading attrs-23.1.0-py3-none-any.whl.metadata (11 kB)
17 Collecting babel==2.14.0 (from -r requirements.txt)
18   Downloading Babel-2.14.0-py3-none-any.whl.metadata (1.3 kB)
19 Collecting beautifulsoup4==4.12.2 (from -r requirements.txt)
20   Downloading beautifulsoup4-4.12.2-py3-none-any.whl.metadata (3.8 kB)
21 Collecting bleach==6.1.0 (from -r requirements.txt)
22   Downloading bleach-6.1.0-py3-none-any.whl.metadata (3.0 kB)
23 Collecting certifi==2024.7.4 (from -r requirements.txt)
24   Downloading certifi-2024.7.4-py3-none-any.whl.metadata (2.2 kB)
25 Collecting charset-normalizer==3.3.2 (from -r requirements.txt)
26   Downloading charset_normalizer-3.3.2-cp312-cp312-macosx_11_0_arm64.whl.metadata (3.4 kB)
27 Collecting defusedxml==0.7.1 (from -r requirements.txt)
28   Downloading defusedxml-0.7.1-py2.py3-none-any.whl.metadata (1.9 kB)
29 Collecting docutils==0.18.1 (from -r requirements.txt)
30   Downloading docutils-0.18.1-py2.py3-none-any.whl.metadata (1.9 kB)
31 Collecting fastjsonschema==2.19.0 (from -r requirements.txt)
32   Downloading fastjsonschema-2.19.0-py3-none-any.whl.metadata (1.9 kB)
```

github actions basics

Composite (reusable) actions

- “Original” way of avoiding duplications and sharing commonly used steps
 - Nowadays entire workflows can also be reused
- Can take inputs and produce outputs
- Very similar structure to workflows
 - Can run several steps
 - Can also call other actions (somewhat limited)
- Can also write actions in javascript or using docker
 - Adapt using: “composite” accordingly
- Place code in *action.yml* on github repository
 - Can choose which version to run via git ref
 - e.g. *key4hep-actions/key4hep-build@main*

```

name: 'Hello World'
description: 'Greet someone'
inputs:
  who-to-greet: # id of input
    description: 'Who to greet'
    required: true
    default: 'World'
outputs:
  random-number:
    description: "Random number"
    value: ${{ steps.random-number-generator.outputs.random-number }}
runs:
  using: "composite"
  steps:
    - name: Set Greeting
      run: echo "Hello $INPUT_WHO_TO_GREET."
      shell: bash
      env:
        INPUT_WHO_TO_GREET: ${{ inputs.who-to-greet }}

    - name: Random Number Generator
      id: random-number-generator
      run: echo "random-number=$(echo $RANDOM)" >> $GITHUB_OUTPUT
      shell: bash

    - name: Set GitHub Path
      run: echo "$GITHUB_ACTION_PATH" >> $GITHUB_PATH
      shell: bash
      env:
        GITHUB_ACTION_PATH: ${{ github.action_path }}

    - name: Run goodbye.sh
      run: goodbye.sh
      shell: bash

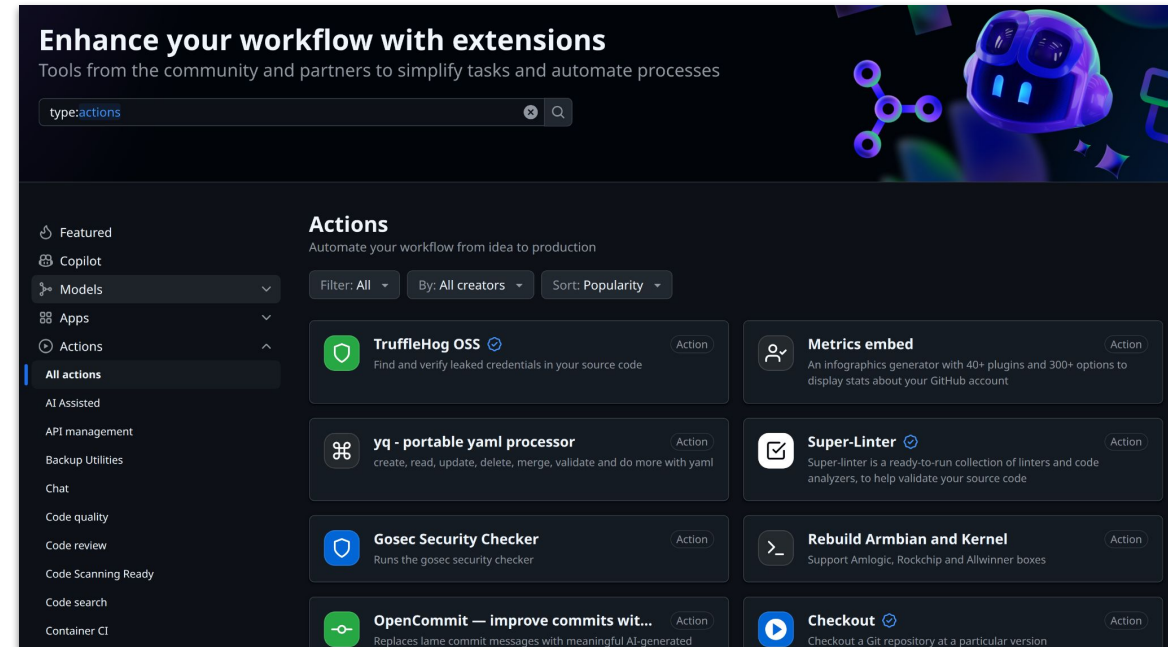
```

Inputs & Outputs

Fantastic actions

... and where to find them

- github.com/marketplace?type=actions - Actions for pretty much everything
- github.com/actions
 - Checking out repositories
 - Setting up dependencies (python, node, ...)
 - Caching data between workflow runs
 - Uploading results as artifacts
 - ...
- github.com/cvfmf-contrib/cvmfs-github-action
 - Install and setup cvmfs on a (ubuntu) github runner
- github.com/key4hep-actions
 - Commonly used actions (and workflows) for Key4hep
 - Some assumptions on how repositories and build system are set up
- github.com/AIDAsoft/run-lcg-view
 - One-stop-shop for running builds and tests inside an LCG environment
- github.com/docker
 - Actions for building and publishing docker images



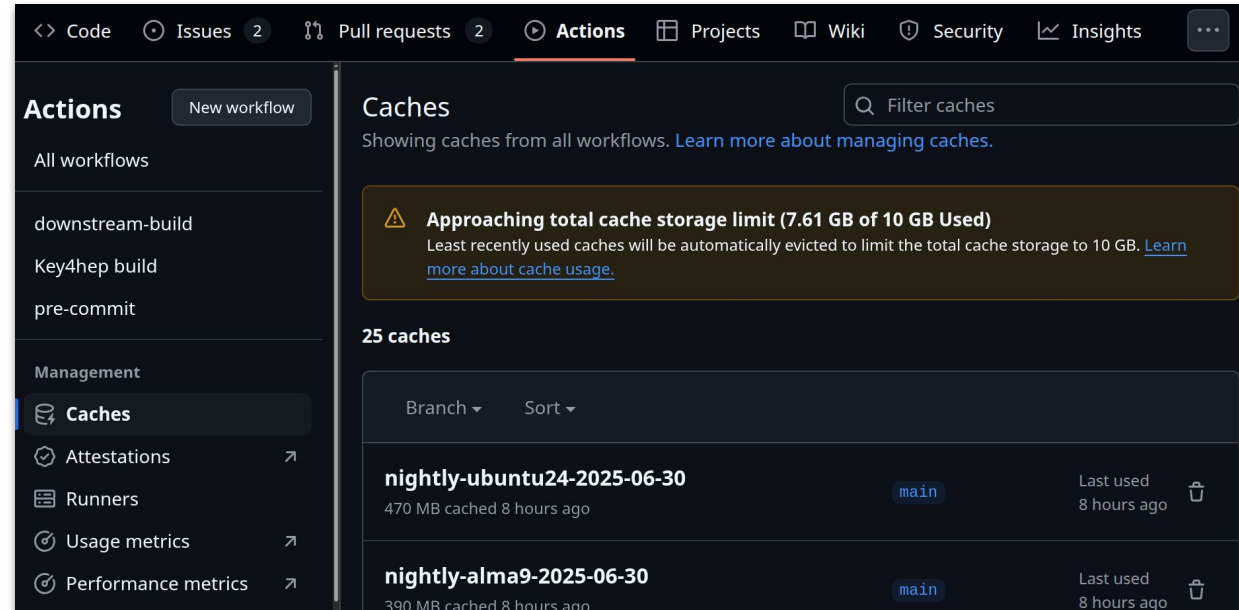
Caching dependencies and other things

Save github some networking costs

- [actions/cache](#) ([documentation](#)) allows to cache and restore arbitrary data
- Provides cache isolation
 - e.g. separate caches for different branches
- Store and restore keys can be different
- 10 GB of cache space available on free github tiers
- Caches evicted after 7 days without access
- Cache management also possible via REST API

```
- uses: actions/cache@v4
  with:
    path: ${{ github.workspace }}/build/ExternalData
    key: ${{ inputs.store-key-base }}
```

[key4hep-actions/cache-external-data](#) caches data (e.g. test inputs) that are fetched via [CMake ExternalData](#)

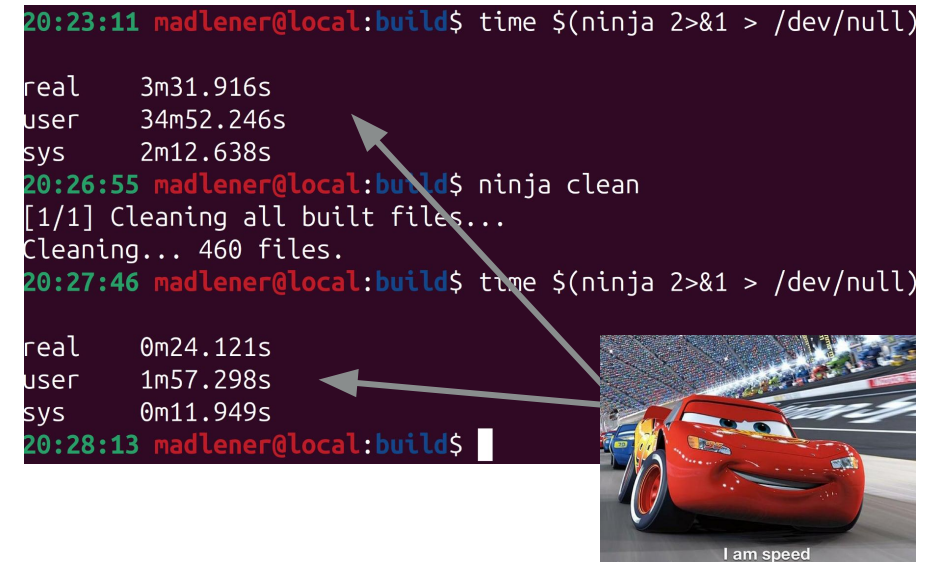


Speeding up CI workflows with caches

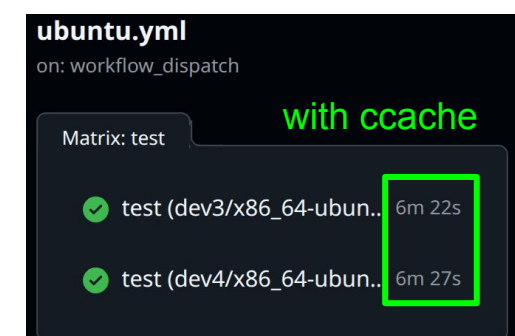
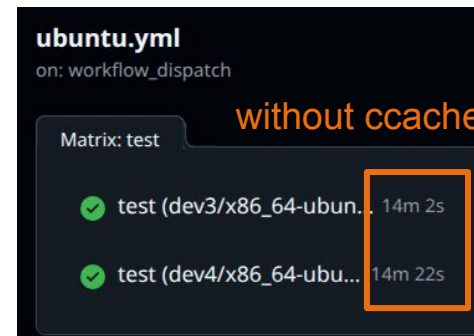
Shoehorning together an (almost) distributed build system

- [ccache](#) (compiler cache) allows to cache compilation results
 - Can dramatically speed up builds
 - Especially “clean builds”
 - Very useful for sharing build artifacts (e.g. across branches or teams)
- Trivial to enable with cmake
 - `-DENABLE_CMAKE_CXX_COMPILER_LAUNCHER=ccache`
- Cache the ccache cache
 - Next build can re-use almost everything
- Tiered caches in gh actions
 - PR runs will find the cache from main branch
- Standard procedure for Key4hep CI
- Probably more environmentally sustainable 🌱
 - Non-trivial considerations for total carbon footprint

```
20:23:11 madlener@local:build$ time $(ninja 2>&1 > /dev/null)
real    3m31.916s
user    34m52.246s
sys     2m12.638s
20:26:55 madlener@local:build$ ninja clean
[1/1] Cleaning all built files...
Cleaning... 460 files.
20:27:46 madlener@local:build$ time $(ninja 2>&1 > /dev/null)
real    0m24.121s
user    1m57.298s
sys     0m11.949s
20:28:13 madlener@local:build$
```



```
- shell: bash
run: echo "NOW=$(date +%Y-%m-%d)" >> $GITHUB_ENV
- uses: actions/cache@v4
  with:
    path: ~/.cache/ccache
    key: ${ inputs.ccache-key }-${ env.NOW }
    restore-keys: |
      ${ inputs.ccache-key }
```



Using dependencies from CVMFS

Let someone else worry about how they get there

- [Cern VM File System](#) - Standard way of distributing software for WLCG
 - Read-only distributed filesystem
 - On-demand loading and caching of binaries
 - Available on many workgroup servers (NAF, lxplus,...)
 - Straightforward to install on local machines
- [LCG releases](#)
 - Curated list of SW releases for the WLCG
 - Used by e.g. ATLAS, LHCb
- [cvmfs-contrib/cvmfs-github-action](#)
 - Install and configure cvmfs
- [AIDASoft/run-lcg-view](#)
 - Prepare container for desired OS with bind-mounted CVMFS
 - Convenience setup for using LCG releases
 - Can use arbitrary SW from CVMFS

```
- uses: actions/checkout@v4
- uses: cvmfs-contrib/github-action-cvmfs@v4
- uses: key4hep/key4hep-actions/cache-external-data@main
- uses: aidasoft/run-lcg-view@v5

with:
  release-platform: LCG_107/x86_64-el9-${{ matrix.compiler }}-opt
  ccache-key: ccache-sanitizers-el9-${{ matrix.compiler }}-${{ matrix.sanitizer }}
```

auto detect OS image and setup script

```
run: |
  echo "::group::Run CMake"
  mkdir -p build
  cd build
  cmake -DCMAKE_BUILD_TYPE=Debug \
    -DUSE_SANITIZER=${{ matrix.sanitizer }} \
    -DCMAKE_CXX_STANDARD=20 \
    -DCMAKE_CXX_COMPILER_LAUNCHER=ccache \
    -DCMAKE_CXX_FLAGS=" -fdiagnostics-color=always " \
    -DUSE_EXTERNAL_CATCH2=OFF \
    -DENABLE_SIO=ON \
    -DENABLE_JULIA=OFF \
    -DENABLE_RNTUPLE=ON \
    -DENABLE_DATASOURCE=ON \
    -G Ninja ..
  echo "::endgroup::"
  echo "::group::Build"
  ninja -k0
  echo "::endgroup::"
```

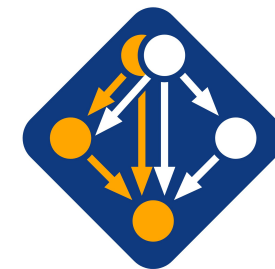
Commands that are run in environment

```
with:
  container: el9
  view-path: /cvmfs/sw-nightlies.hsf.org/key4hep
```

specify image and base manually

The spack package manager

Build your stack from scratch



- [spack](#) is a platform and OS independent package manager
 - Supports multiple languages, build systems, ...
- Emphasis on dealing with multiple configurations of the same package
 - Versions, build flags, features, ...
 - Resulting software stacks are consistent by construction
- Builds all packages from source
 - spack package description in python
 - Can re-use packages from underlying system
- Originally developed by HPC community
- Also used by Key4hep
 - github.com/key4hep/key4hep-spack package repository
- Supports a *build cache* to reuse packages already built
 - **Can use OCI registries as build caches**

```
class Sio(CMakePackage):
    homepage = "https://github.com/iLCSoft/SIO"
    git = "https://github.com/iLCSoft/SIO.git"

    maintainers("vvolkl", "tmadlener", "jmcarcell")

    license("BSD-3-Clause")

    version("master", branch="master")
    version("0.2", sha256="416c93402e7314b7aadedba8e7f9e0d4b0b4f4e34ce26285b04cebb505ecfab2")
    version("0.1", sha256="0407c0daae53660c0562f9302a220f72ab51547050cd9fe9113b995804ab4b4")
    version("0.0.4", sha256="72e96e6a1cc8dd3641d3e2bb9876e75bf6af8074e1617220da9e52df522ef5c0")
    version("0.0.3", sha256="4c8b9c08480fb53cd10abb0e1260071a8c3f68d06a8acfd373f6560a916155cc")
    version("0.0.2", sha256="e4cd2aeaeaa23c1da2c20c5c08a9b72a31b16b7a8f5aa6d480dcd561ef667657")

    variant(
        "builtin_zlib",
        default=True,
        description="Use and statically link against a builtin version of zlib",
    )
    variant(
        "cxxstd",
        default="17",
        values=("11", "14", "17", "20"),
        multi=False,
        description="Use the specified C++ standard when building.",
    )

    depends_on("c", type="build") # generated
    depends_on("cxx", type="build") # generated

    depends_on("zlib-api", when="~builtin_zlib")
```

Example: building the muon collider software stack

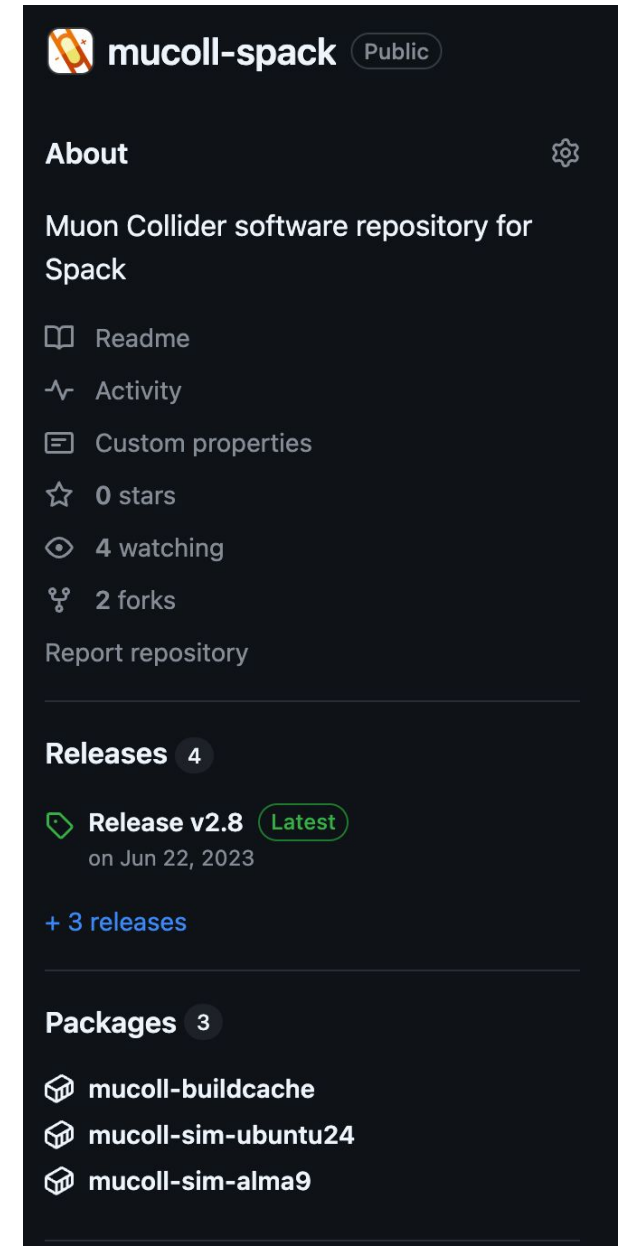
Last year the IMCC (International Muon Collider Collaboration) set up a task force tasked with rationalising the development of software for the experiments.

Most important outcomes for this talk:

- Adoption of Key4hep
- Need to **build and deploy software releases with docker**

IMCC software task force recommendations

- IMCC should focus on a single avenue for software distribution
 - docker images were selected as the most modern/portable solution
 - optionally replicated to cvmfs via unpacked
- **Images should be automatically built using CI tools**
- **Images should be centrally published in a single container repository**
 - **ghcr.io feels like the most natural choice so that containers can be directly linked to release notes/pages**



Can we do it with ~no resources?

Free github runners

Substantial resources are available for both public and private repositories [\[overview on github\]](#)

However, they come with some limitations:

- Runtime of **each workflow job** limited to 6h
- VM has total of ~72 GB disk
- Throttling if you use too many resources (never hit this limitation so far)

| Virtual Machine | Processor (CPU) | Memory (RAM) | Storage (SSD) | Architecture | Workflow label |
|--------------------------|-----------------|--------------|---------------|--------------|---|
| Linux | 4 | 16 GB | 14 GB | x64 | <code>ubuntu-latest</code> , <code>ubuntu-24.04</code> , <code>ubuntu-22.04</code> |
| Windows | 4 | 16 GB | 14 GB | x64 | <code>windows-latest</code> , <code>windows-2025</code> , <code>windows-2022</code> , <code>windows-2019</code> |
| Linux [Public preview] | 4 | 16 GB | 14 GB | arm64 | <code>ubuntu-24.04-arm</code> , <code>ubuntu-22.04-arm</code> |
| Windows [Public preview] | 4 | 16 GB | 14 GB | arm64 | <code>windows-11-arm</code> |
| macOS | 4 | 14 GB | 14 GB | Intel | <code>macos-13</code> |
| macOS | 3 (M1) | 7 GB | 14 GB | arm64 | <code>macos-latest</code> , <code>macos-14</code> , <code>macos-15</code> |

Free github runners

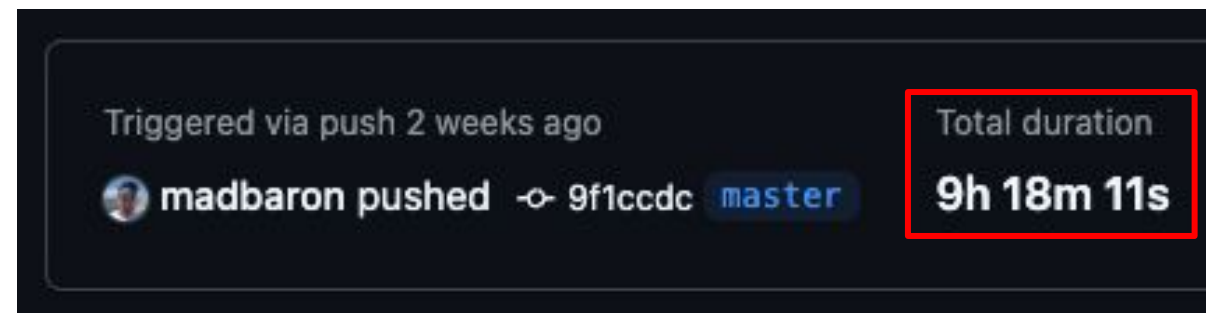
No free lunch

The workflow using spack requires by default to rebuild the **whole spack recipe**

- This means compiling a substantial amount of “external” packages that are not directly part of the software stack
- Compiling these takes 60-70% of the total time (and **too long overall**)

These packages are updated relatively rarely

Also the software stack consists of ~15 GB of software, enough to **deplete the disk space of the runner** while compiling



Solving the space issue

Who asked for this anyway?

The default github runners come with a substantial amount of software pre-installed

- We only need docker

Actions are available to delete unused features

- can reclaim up to 38GB of disk

```
85 Removing /usr/share/gl_*  
86 FreeUp Space: 466.65 MB  
87 Time Elapsed: 1 seconds  
88 Removing /usr/share/glade*  
89 FreeUp Space: 0 MB  
90 Time Elapsed: 0 seconds  
91 Removing /usr/local/lib/node_modules  
92 FreeUp Space: 649.99 MB  
93 Time Elapsed: 8 seconds  
94 Removing /usr/local/share/chromium  
95 FreeUp Space: 572.71 MB  
96 Time Elapsed: 0 seconds  
97 Removing /usr/local/share/powershell  
98 FreeUp Space: 1230.38 MB  
99 Time Elapsed: 0 seconds  
100 Total Free Space: 37700.88 MB
```

```
steps:  
- name: Free Disk Space (Ubuntu)  
  uses: endersonmenezes/free-disk-space@v2  
  with:  
    remove_android: true  
    remove_dotnet: true  
    remove_haskell: true  
    remove_tool_cache: true  
    remove_swap: true  
    remove_packages: "azure-cli google-cloud-cli microsoft-edge-stable google-chrome-stable firefox postgresql* temurin-* *llvm* mysql* dotnet-sdk-*"   
    remove_packages_one_command: true  
    remove_folders: "/usr/share/swift /usr/share/miniconda /usr/share/az* /usr/share/glade* /usr/local/lib/node_modules /usr/local/share/chromium /usr/local/share/powershell"  
    testing: false
```

Solving the timeout issue

xkcd.com/303 anyone?

So we are left with the compilation time taking too long for the github runners

A few options to solve the issue:

Use dedicated self-hosted runners

Break the compilation into sub-steps

Ship partial workflows to another git instance (i.e. CERN/DESY)

Use OCI build caches

Solving the timeout issue

xkcd.com/303 anyone?

So we are left with the compilation time taking too long for the github runners

A few options to solve the issue:

Use dedicated self-hosted runners

Ship partial workflows to another git instance (i.e. CERN/DESY)

Excellent documentation available [on github](#)

Easy to set up and run

- the runner polls github for jobs and sends back the results - permissions/authentication is simple

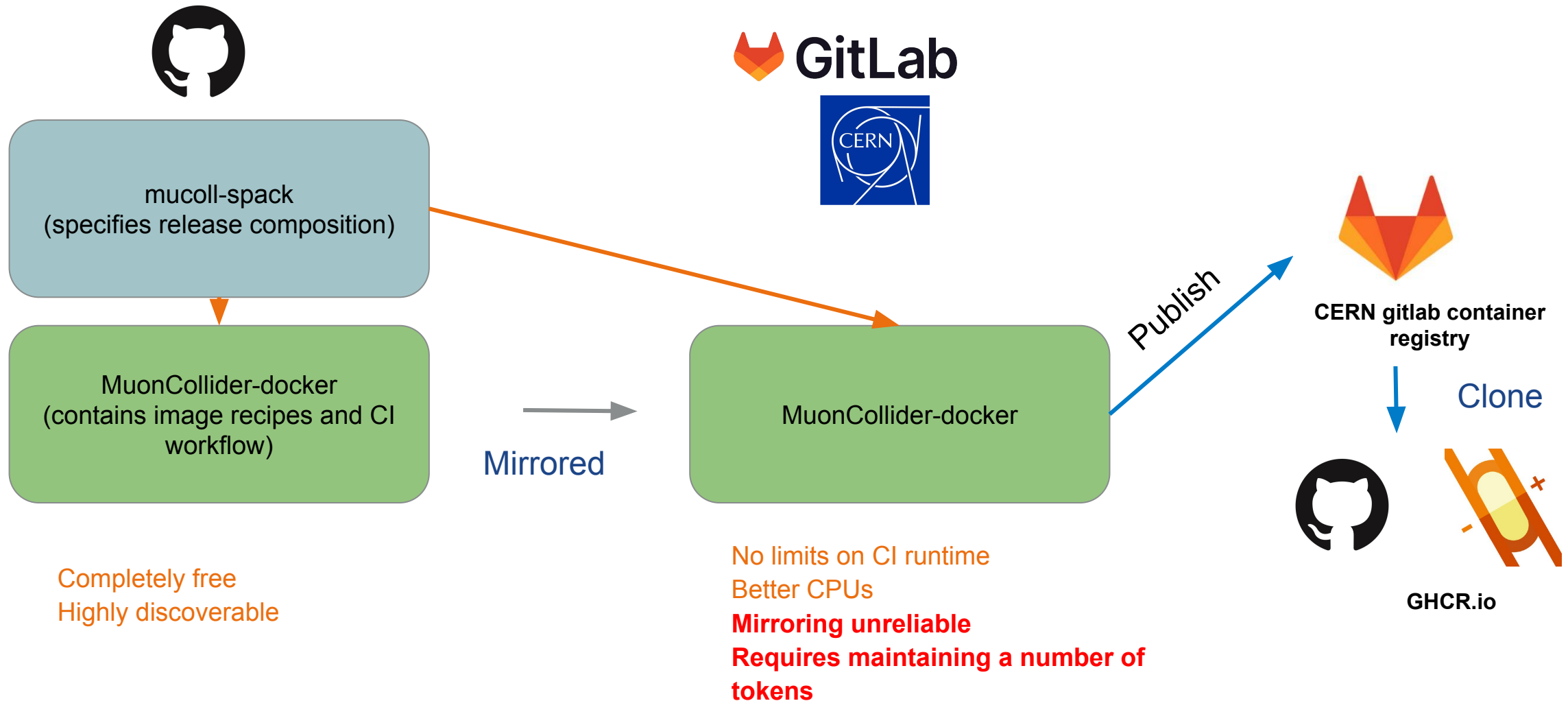
Fun exercise, tried with:

- Desktop in the office
- VM at CERN

Works, but **doesn't scale well and requires some level of maintenance**

Github to gitlab synchronisation?

We have fast machines at home



Key4hep-dev-externals

Software stacks all the way down

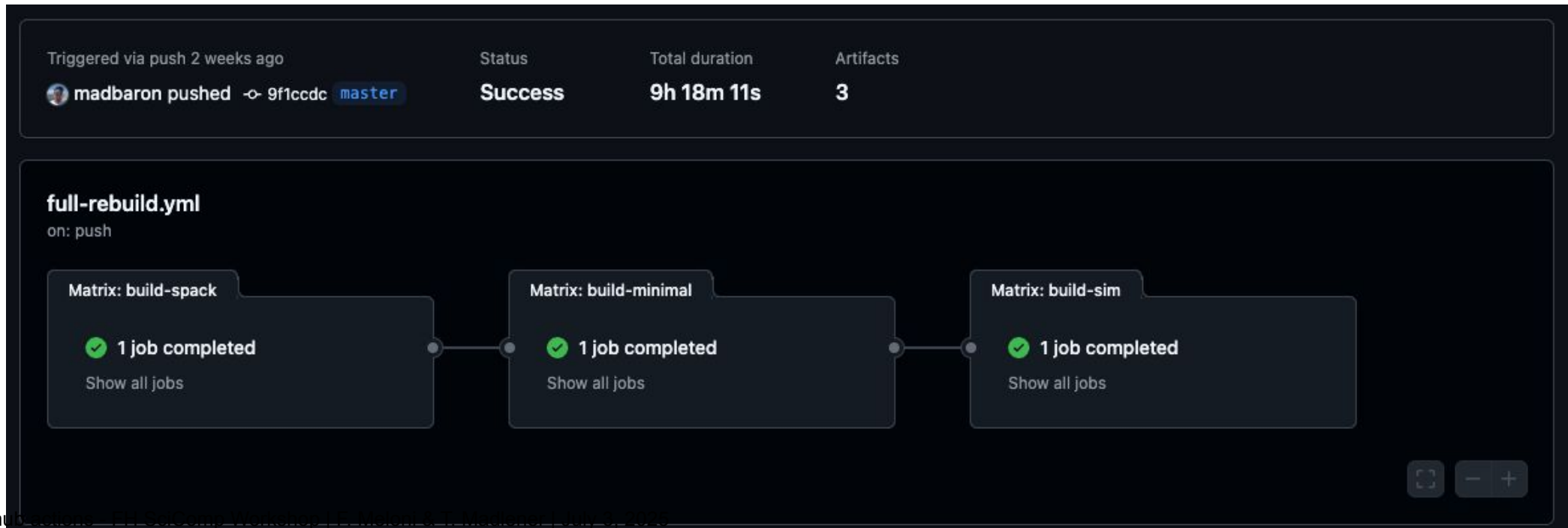
Developed an “intermediate” stack within key4hep software stack, bundling together all external dependencies: **key4hep-dev-externals**

Build experiment-specific stacks on top of it!

Useful for mucoll, LUXE, ...

Factorised build workflow in three steps, each one running below the 6h limit

- Intermediate steps publish docker images that are re-used in the following



Using OCI build caches

Spack supports using OCI build caches to speed up long compilation processes.

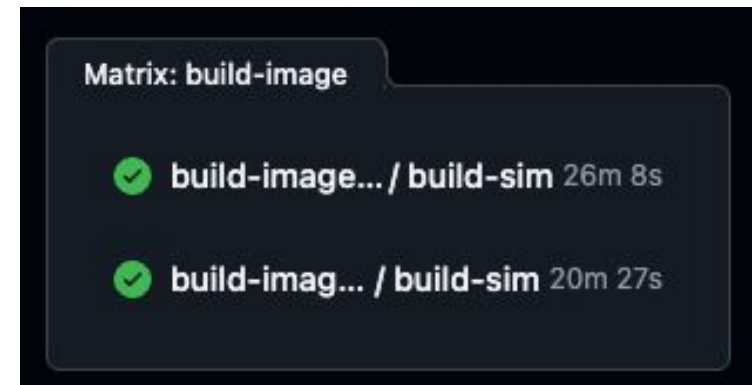
- These are stored on the github docker registry as docker images
- **No cache size or expiration limits!**

```
- name: Build and Push
  id: docker_build
  uses: docker/build-push-action@v6
  with:
    push: true
    context: ./Docker
    file: ./Docker/Dockerfile-sim
    build-args: |
      OS=${{ inputs.os }}
      GITHUB_REPOSITORY=${{ github.repository }}
      MUCOLL_SHA=${{ github.sha }}
      SPACK_BUILD_CACHE=oci://ghcr.io/${{ github.repository_owner }}/mucoll-buildcache
      OCI_USERNAME=${{ github.actor }}
    secrets: |
      "ocipass=${{ secrets.GITHUB_TOKEN }}"
  tags: ${ steps.meta.outputs.tags }
  labels: ${ steps.meta.outputs.labels }
```

```
spack repo add --scope system ${SPACK_ROOT}/var/mucoll-spack && \
if [ -n "${SPACK_BUILD_CACHE}" ]; then \
  spack mirror add --oci-username "${OCI_USERNAME}" --oci-password "${OCI_PASSWORD}" --unsigned --autopush local-buildcache "${SPACK_BUILD_CACHE}"; \
fi
```

Total workflow time: ~20 minutes (from 9h!)

Still want to have each part of the workflow to run in less than 6 hours, to avoid failures if starting from scratch



Don't leak your secrets

Not even to your friends

Several of these workflows require a number of authentication steps

Never store your access credentials or tokens in plain view

- If you (accidentally) do, you should immediately consider the credential compromised (bots **will** find them!!)

Github/lab secrets are there for you!

- More convoluted for docker_build
- Secrets are mounted as files in the image

Example docker workflow

In workflow file

```
- name: Build and Push
  id: docker_build
  uses: docker/build-push-action@v6
  with:
    push: true
    context: ./Docker/${{inputs.os}}
    file: ./Docker/${{inputs.os}}/Dockerfile-externals
    build-args: |
      GITHUB_REPOSITORY=${{ github.repository }}
      COMMIT_SHA=${{ github.sha }}
      SPACK_COMMIT=${{ steps.getref.outputs.spack_ref }}
      SPACK_BUILDCACHE=oci://ghcr.io/${{ github.repository_owner }}/spack-buildcache
      OCI_USERNAME=${{ github.actor }}
    secrets: |
      "ocipass=${{ secrets.GITHUB_TOKEN }}"
    tags: ${ steps.meta.outputs.tags }
    labels: ${ steps.meta.outputs.labels }
```

In Dockerfile

```
RUN --mount=type=secret,id=ocipass \
  OCI_PASSWORD=$(cat /run/secrets/ocipass) && \
  . /opt/setup_spack.sh && \
  spack repo add --scope system ${SPACK_ROOT}/var/mucoll-spack && \
  if [ -n "${SPACK_BUILDCACHE}" ]; then \
    spack mirror add --oci-username "${OCI_USERNAME}" --oci-password "${OCI_PASSWORD}" \
  fi
```

Summary

- CI on github repositories is easy to setup
- Large ecosystem of existing actions
- Developed a few actions that are generally useful for HEP SW in Key4hep context
- Surprising amount of free computing resources on github runners - can deal with even relatively large and complex projects
- Some creativity required to work around remaining limitations
- Caches are your friends
 - Even better if they don't count towards some quota ;)
- Keep your secrets safe!

Thanks

Tips & Tricks

Things we partially learned the hard way

- Logs are (almost) everything you get in case of failure
 - Read them carefully ;)
 - Consider increasing verbosity of tools
 - Use `::group:<group-name>` and `::endgroup::` for maintaining sanity in success cases
- You can run [steps even if previous steps failed](#)
- Consider running CI inside a container
 - Reproducible even if github runner VM changes
 - Godsend when you need to debug some more involved issues
- Take care to not leak secrets (API keys, Tokens, ...)
 - Store them as secrets via github settings
 - Access via `${{ secrets.<SECRET_NAME> }}`
- Consider cancelling concurrent runs
 - Helps in avoiding throttling of workflow runs on free tier

```
echo "::group::Build Catch2"
cd $STARTDIR/catch2
mkdir build && cd build
cmake -DCMAKE_CXX_STANDARD=20 -DCMAKE
ninja -k0 install
export CMAKE_PREFIX_PATH=$STARTDIR/ca
echo "::endgroup::"
```

| | |
|------|--------------------------|
| 174 | ► Build Catch2 |
| 523 | ► Build podio |
| 616 | ► Test and install podio |
| 781 | ► Build and test EDM4hep |
| 1123 | ► ccache statistics |

```
steps:
  ...
  - name: The job has failed
    if: ${{ failure() }}
```

```
coverity-project-token: ${{ secrets.PODIO_COVERITY_TOKEN }}
github-pat: ${{ secrets.READ_COVERITY_IMAGE }}
```

```
coverity-project-token: ***
github-pat: ***
```

```
concurrency:
  group: ${{ github.workflow }}-${{ github.ref }}
  cancel-in-progress: true
```


Links to the various screenshots

- <https://github.com/key4hep/key4hep-spack/blob/main/.github/workflows/image-template.yaml>
 - Docker image build and spack build cache usage
- <https://github.com/AIDASoft/podio/blob/master/.github/workflows/edm4hep.yaml>
 - Full example of workflow for building dependencies along the way
- <https://github.com/AIDASoft/podio/blob/master/.github/workflows/key4hep.yml>
 - Example for using run-lcg-view without an LCG release
- <https://github.com/key4hep/key4hep-doc/blob/main/.github/workflows/publish.yml>
 - Example for building and deploying documentation via github pages via github actions