SPACE

CENTRE OF EXCELLENCE FOR HPC
ASTROPHYSICAL APPLICATIONS

USM Code Coffee
A beginers guide to Continuous Integration

Geray Karademir, LMU

Co-funded by
the European Union

Funded by the European Union. This work has received funding from the European High Performance Computing Joint Undertaking (JU) and Belgium, Czech Republic, France, Germany, Greece, Italy, Norway, and Spain under grant agreement No 101093441.

EuroHPC
Joint Undertaking

What is Continous Integration?

Continuous Integration (CI) is the practice of merging all your code changes to the mainline of the code early and often. This merge is combined with an automatic testing for each change.

Why is it important?

- By merging frequently and triggering automatic testing the potential for code conflicts is reduced.

- Errors and security issues can be identified early in the development process.

- Fixing potential bugs is done right after their implementation which reduces the numbers of changes to investigate.

What is Continuous Delivery?

- Continuous Delivery (CD) is the practice to release software in short cycles. In this process the developers ensure that the software can be reliably released at any time and following a pipeline through a "production-like environment", without having to do so manually.

- The approach is aimed to reduce the cost, time, and risk of delivering changes by allowing for more incremental updates.

What is a CI/CD Pipeline?

The CI/CD pipeline automates the process of creating, testing and deploying an application.

Integrating an automated CI/CD pipeline significantly reduces the risk of errors in the deployment process and ensures that bugs are caught early.

Lets get started!

Requirements:

- A GitLab repository with a code.

- A system to run the tests:
  - This system must be able to run the code (libraries, etc.)

- Test cases:
  - Compilation test
  - Unit tests
  - …

Outline of setting up a CI pipeline

1. Setup workers
   - Installing GitLab-runners on system
   - Create runners
   - Register runners

2. Create process script
   - Create pipeline
   - Shedule pipeline
   - Other options

3. Run jobs

Create runners

What is a runner?

- A runner is an application that works with GitLab CI/CD to run jobs in a pipeline.

- It allows you to run jobs on your own machines via the GitLab Runner service, by connecting them to GitLab.

How to install gitlab-runner on system

- Before creating the runner, you have to make sure that the gitlab-runner software is installed on your system: https://docs.gitlab.com/runner/install/

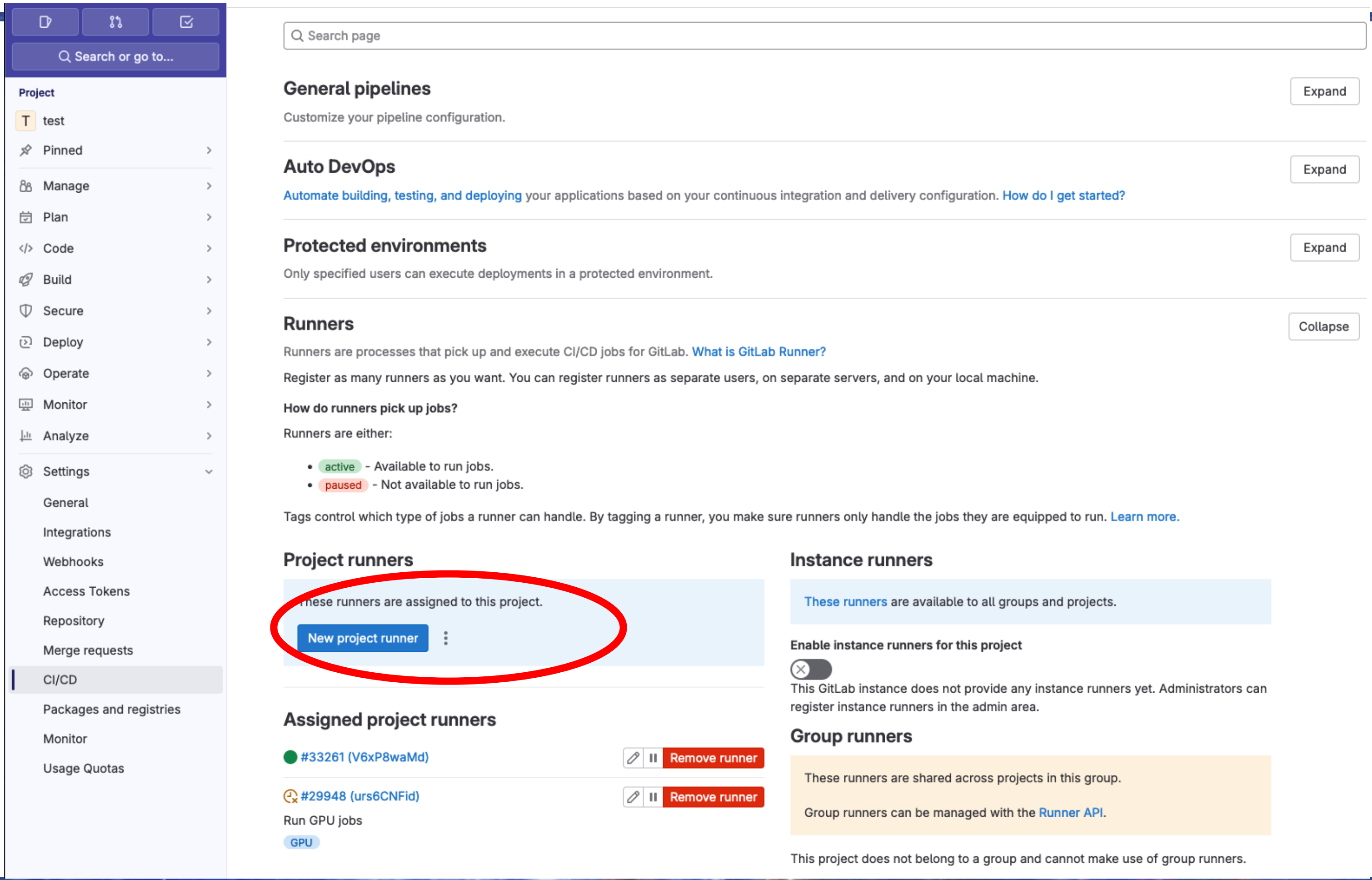
- Note: Always install files from link above and not with tools like “apt” as version will be highly outdated!

- It is important that the GitLab version (can be found at https://gitlab.LRZ.com/help/) is the same as the runner version.

Create runner

- Create runner in settings of project repo

- Runner type:
  - Project runners: runners exclusively avaliiable to your project.
  - Instance runners: runners shared across multiple projects.



General pipelines

Customize your pipeline configuration.

Auto DevOps

Automate building, testing, and deploying your applications based on your continuous integration and delivery configuration. How do I get started?

Protected environments

Only specified users can execute deployments in a protected environment.

Runners

Runners are processes that pick up and execute CI/CD jobs for GitLab. What is GitLab Runner?

Register as many runners as you want. You can register runners as separate users, on separate servers, and on your local machine.

How do runners pick up jobs?

Runners are either:

- active - Available to run jobs.
- paused - Not available to run jobs.

Tags control which type of jobs a runner can handle. By tagging a runner, you make sure runners only handle the jobs they are equipped to run. Learn more.

Project runners

These runners are assigned to this project.

New project runner

Assigned project runners

#33261 (V6xP8waMd) Remove runner

#29948 (urs6CNFid) Remove runner

Run GPU jobs

GPU

Instance runners

These runners are available to all groups and projects.

Enable instance runners for this project

This GitLab instance does not provide any instance runners yet. Administrators can register instance runners in the admin area.

Group runners

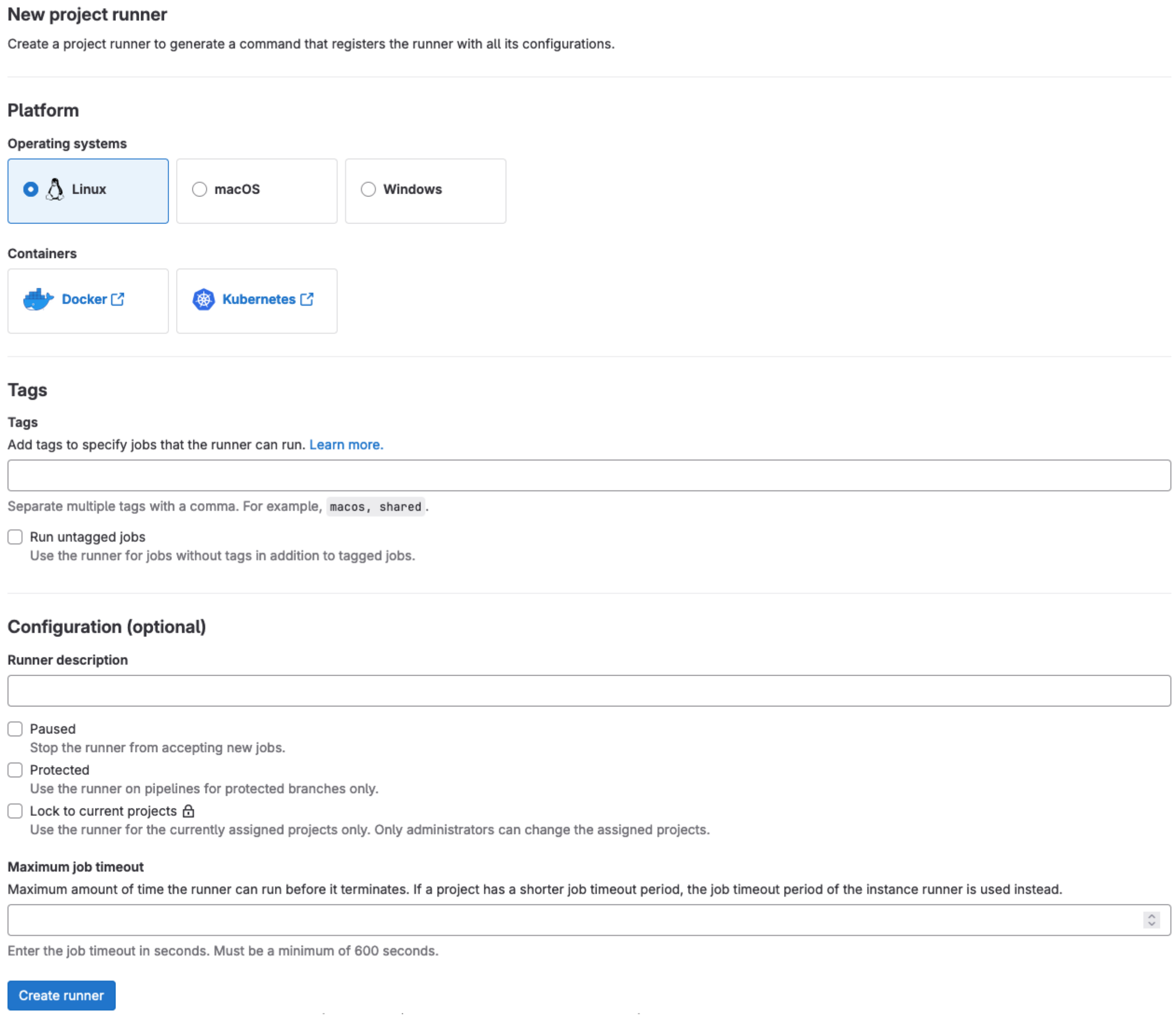
These runners are shared across projects in this group.

Group runners can be managed with the Runner API.

This project does not belong to a group and cannot make use of group runners.

Create runner

- Tags:
  - Allows to link specific runners to specific jobs, e.g. GPU jobs to a runner which has access to GPUs.
  - If no tags are provided, you have to allow the runner to run jobs without tags.
- Additional settings in configuration.



New project runner

Create a project runner to generate a command that registers the runner with all its configurations.

Platform

Operating systems

● Linux ○ macOS ○ Windows

Containers

Docker Kubernetes

Tags

Tags

Add tags to specify jobs that the runner can run. Learn more.

Separate multiple tags with a comma. For example, `macos, shared`.

□ Run untagged jobs
Use the runner for jobs without tags in addition to tagged jobs.

Configuration (optional)

Runner description

□ Paused
Stop the runner from accepting new jobs.

□ Protected
Use the runner on pipelines for protected branches only.

□ Lock to current projects
Use the runner for the currently assigned projects only. Only administrators can change the assigned projects.

Maximum job timeout

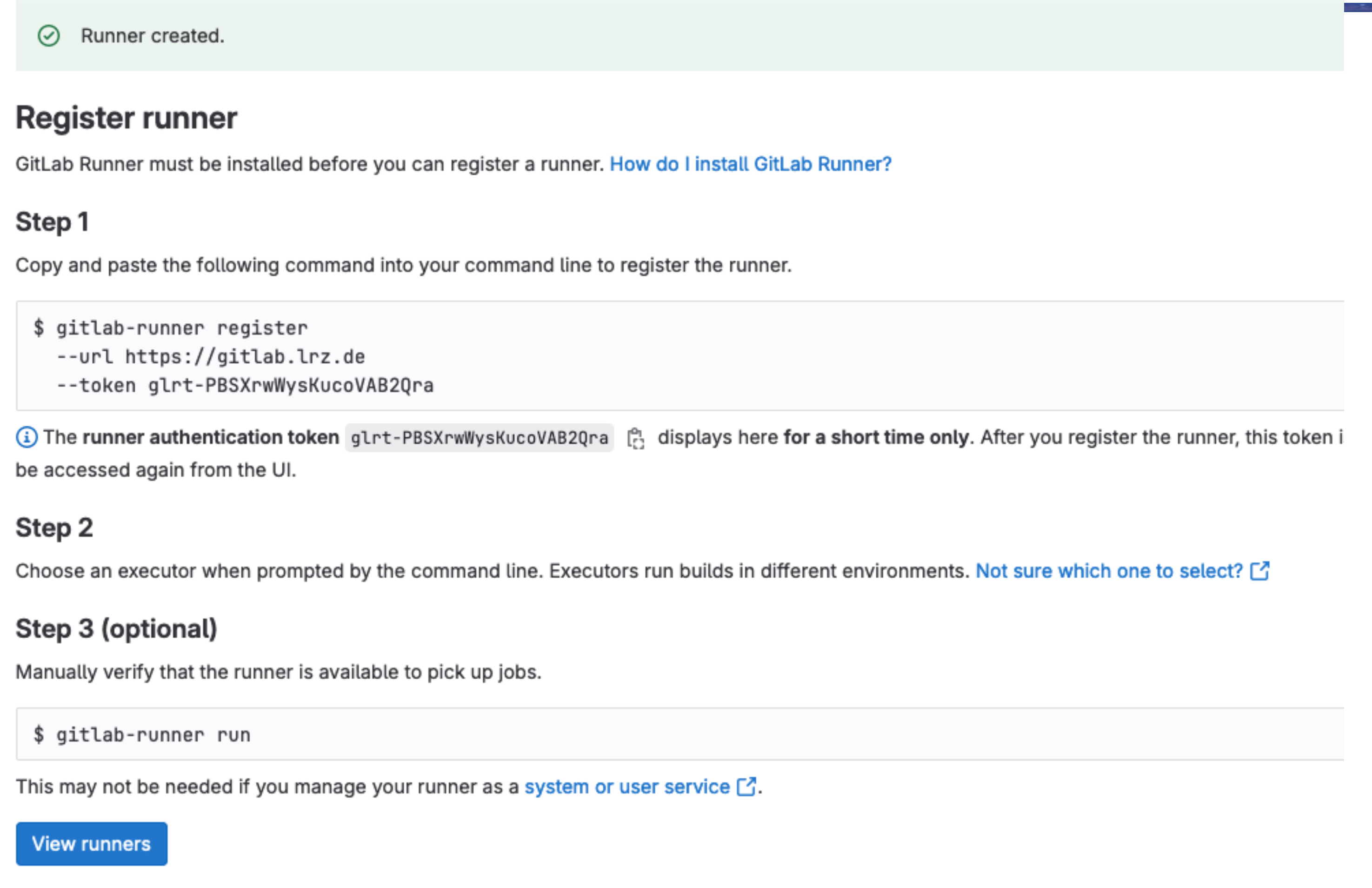
Maximum amount of time the runner can run before it terminates. If a project has a shorter job timeout period, the job timeout period of the instance runner is used instead.

Enter the job timeout in seconds. Must be a minimum of 600 seconds.

Register runner

- When registering runner:
  - Define executor:
    - SSH
    - Shell
    - Docker (Autoscaler, Machine)
    - Parallels
    - VirtualBox
    - Kubernetes
    - Instance
    - Custom

- More details: https://docs.gitlab.com/runner/executors/index.html



✓ Runner created.

Register runner

GitLab Runner must be installed before you can register a runner. How do I install GitLab Runner?

Step 1

Copy and paste the following command into your command line to register the runner.

```
$ gitlab-runner register
  --url https://gitlab.lrz.de
  --token glrt-PBSXrwWysKucoVAB2Qra
```

ⓘ The **runner authentication token** `glrt-PBSXrwWysKucoVAB2Qra` displays here **for a short time only**. After you register the runner, this token i be accessed again from the UI.

Step 2

Choose an executor when prompted by the command line. Executors run builds in different environments. Not sure which one to select?

Step 3 (optional)

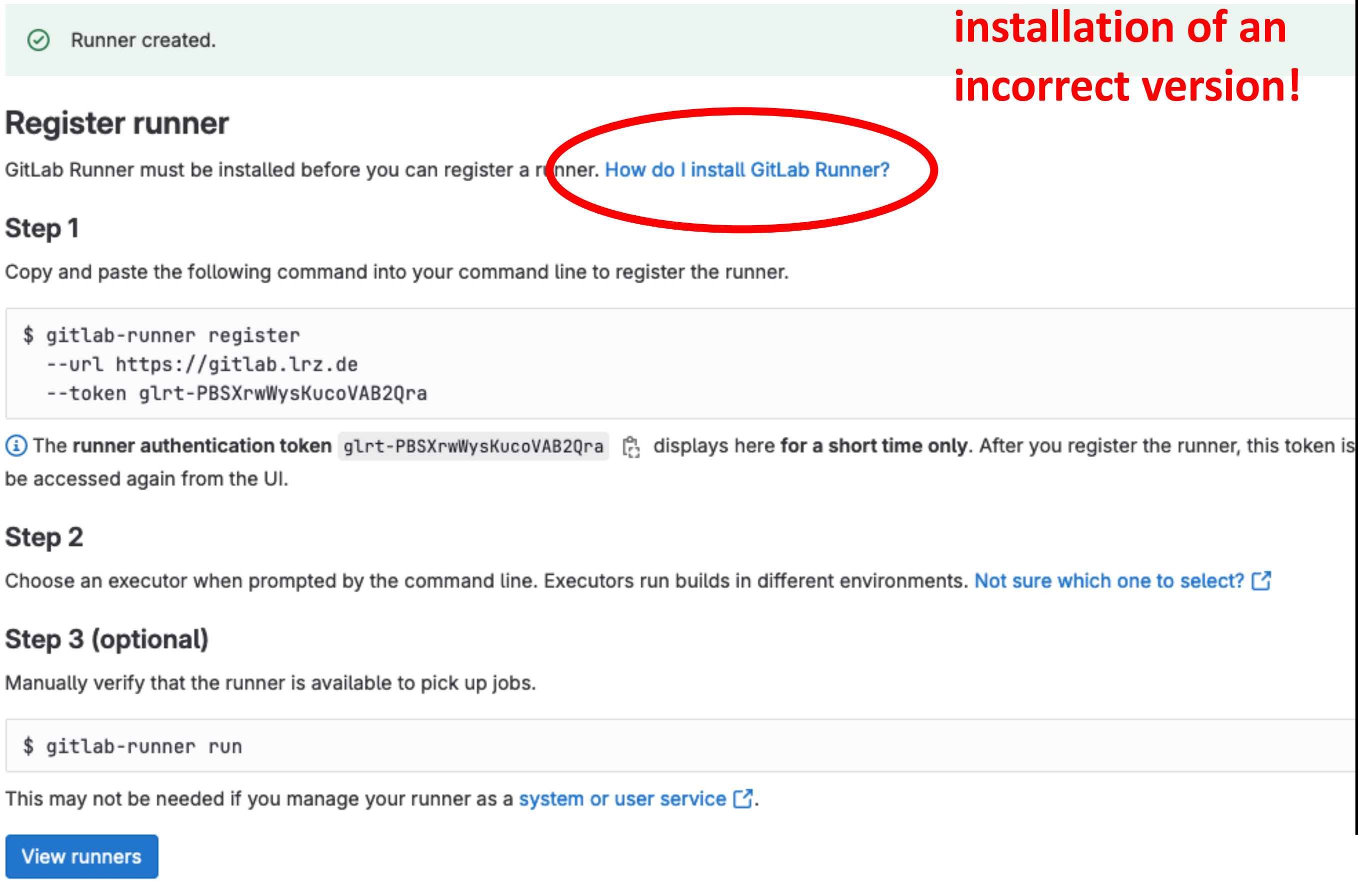
Manually verify that the runner is available to pick up jobs.

```
$ gitlab-runner run
```

This may not be needed if you manage your runner as a system or user service.

Create runner

Avoid doing so: this might lead to the installation of an incorrect version!



Runner created.

Register runner

GitLab Runner must be installed before you can register a runner. How do I install GitLab Runner?

Step 1

Copy and paste the following command into your command line to register the runner.

```
$ gitlab-runner register
  --url https://gitlab.lrz.de
  --token glrt-PBSXrwWysKucoVAB2Qra
```

The **runner authentication token** `glrt-PBSXrwWysKucoVAB2Qra` displays here **for a short time only**. After you register the runner, this token is s[...] be accessed again from the UI.

Step 2

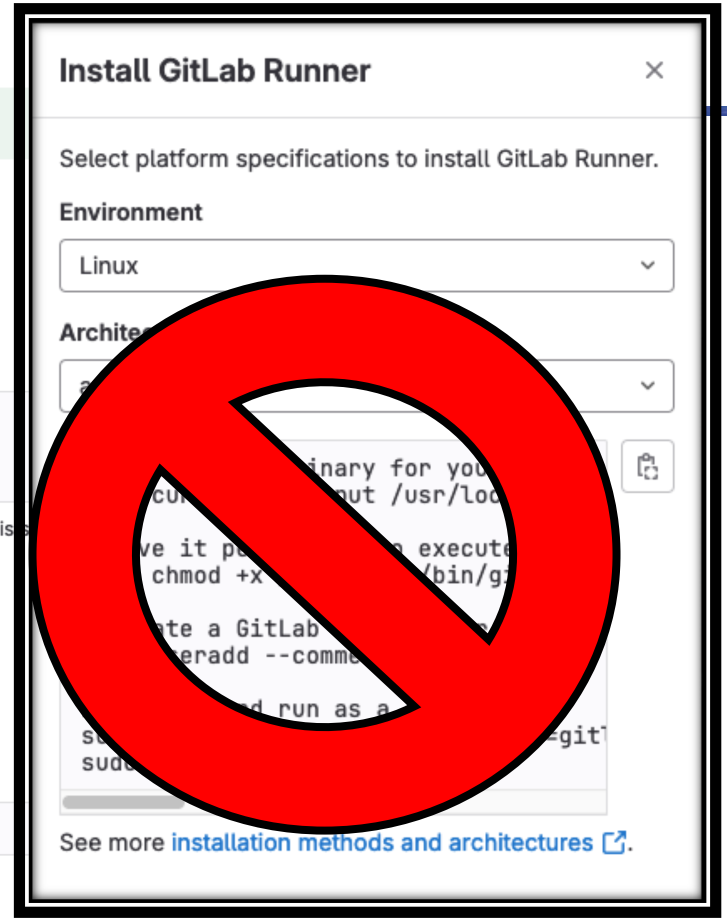
Choose an executor when prompted by the command line. Executors run builds in different environments. Not sure which one to select?

Step 3 (optional)

Manually verify that the runner is available to pick up jobs.

```
$ gitlab-runner run
```

This may not be needed if you manage your runner as a system or user service.



Install GitLab Runner

Select platform specifications to install GitLab Runner.

Environment: Linux

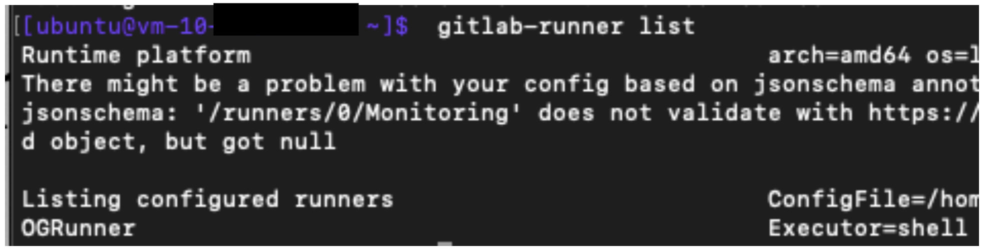
Archite

```
                  inary for you
cu                put /usr/loc

ve it p             execute
chmod +x            /bin/g

te a GitLab
eradd --comm

    d run as a

s                           git
sud
```

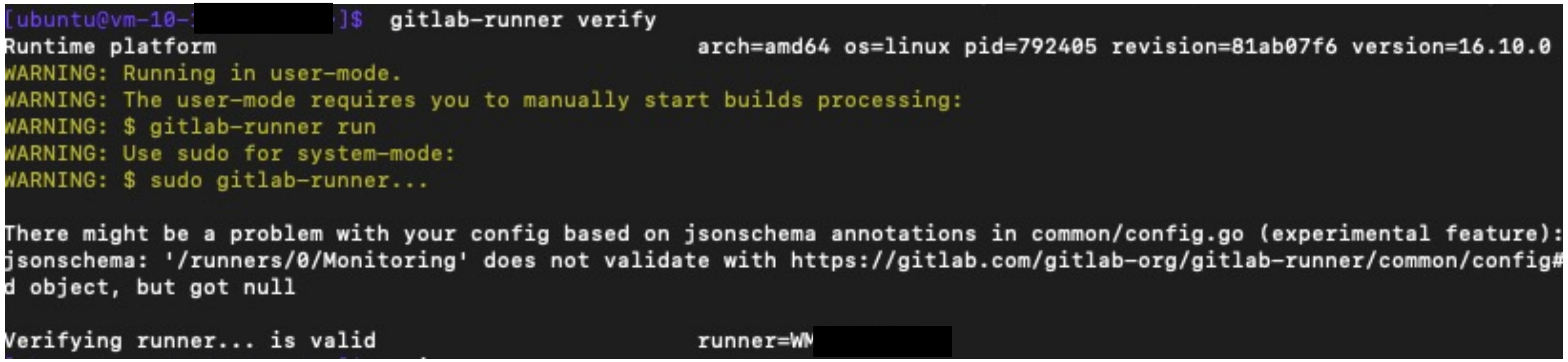
See more installation methods and architectures.

Check if we were succesfull:

- On system:
  - Gitlab-runner list
  - Gitlab-runner verify



```
[[ubuntu@vm-10-[redacted] ~]$  gitlab-runner list
Runtime platform                                    arch=amd64 os=l
There might be a problem with your config based on jsonschema annot
jsonschema: '/runners/0/Monitoring' does not validate with https://
d object, but got null

Listing configured runners                          ConfigFile=/hom
OGRunner                                            Executor=shell
```

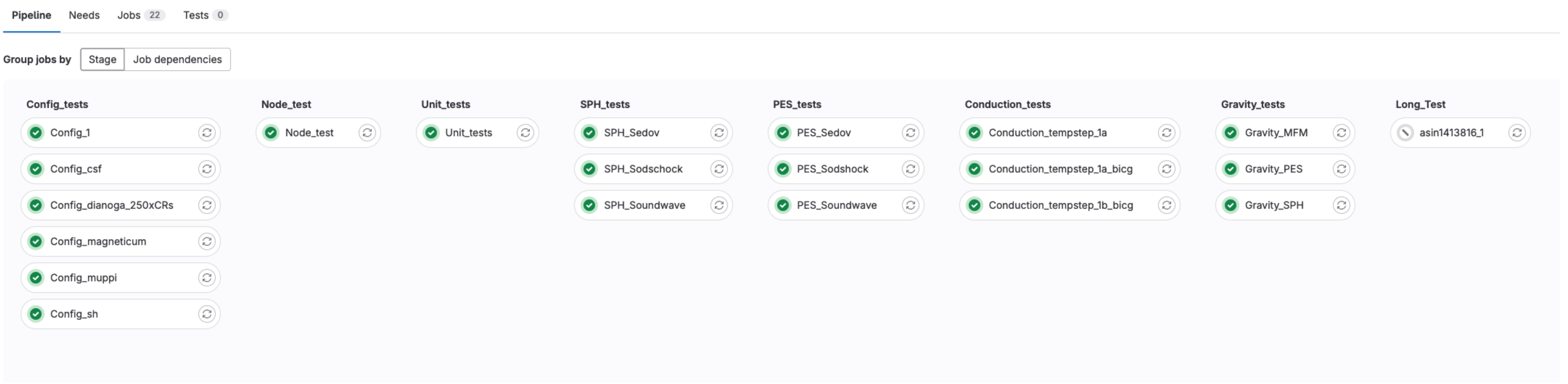


```
[ubuntu@vm-10-[redacted]]$  gitlab-runner verify
Runtime platform                                    arch=amd64 os=linux pid=792405 revision=81ab07f6 version=16.10.0
WARNING: Running in user-mode.
WARNING: The user-mode requires you to manually start builds processing:
WARNING: $ gitlab-runner run
WARNING: Use sudo for system-mode:
WARNING: $ sudo gitlab-runner...

There might be a problem with your config based on jsonschema annotations in common/config.go (experimental feature):
jsonschema: '/runners/0/Monitoring' does not validate with https://gitlab.com/gitlab-org/gitlab-runner/common/config#
d object, but got null

Verifying runner... is valid                        runner=WM[redacted]
```

A simple example hands on: setting up a pipeline!

The CI pipeline of OpenGADGET3:

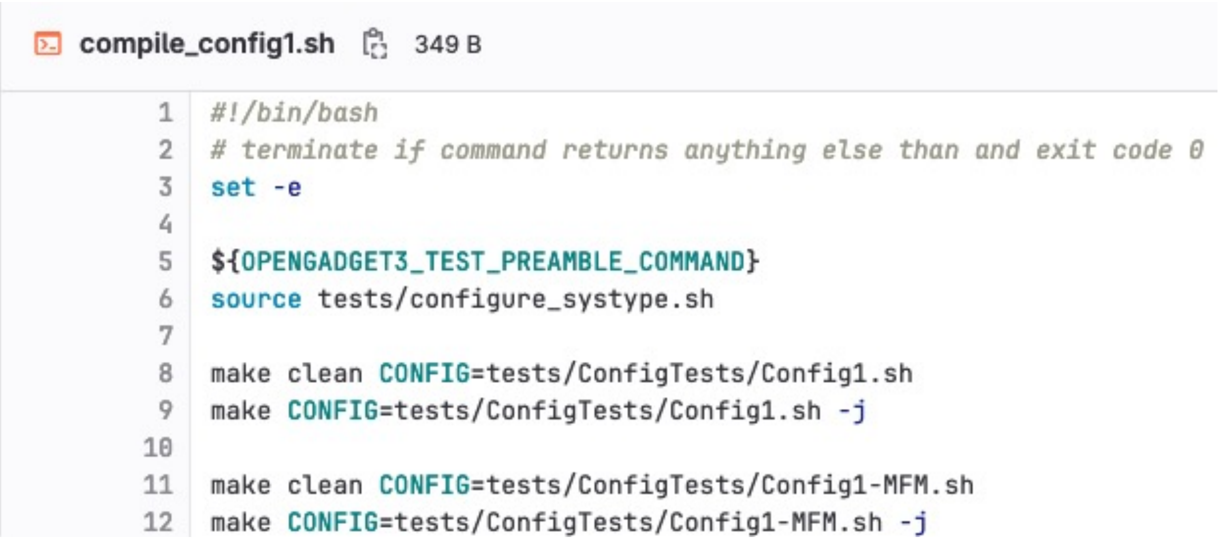
The full pipeline:



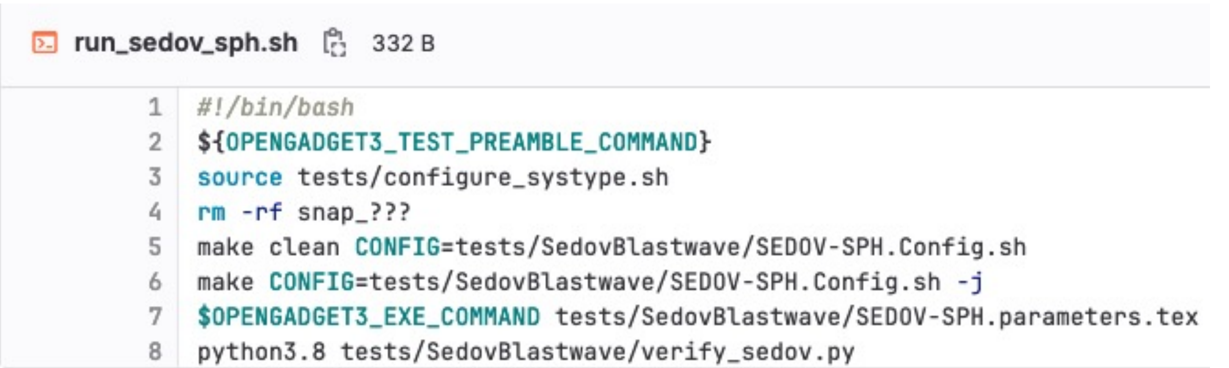
Config_tests	Node_test	Unit_tests	SPH_tests	PES_tests	Conduction_tests	Gravity_tests	Long_Test
✓ Config_1	✓ Node_test	✓ Unit_tests	✓ SPH_Sedov	✓ PES_Sedov	✓ Conduction_tempstep_1a	✓ Gravity_MFM	⊘ asin1413816_1
✓ Config_csf			✓ SPH_Sodschock	✓ PES_Sodshock	✓ Conduction_tempstep_1a_bicg	✓ Gravity_PES	
✓ Config_dianoga_250xCRs			✓ SPH_Soundwave	✓ PES_Soundwave	✓ Conduction_tempstep_1b_bicg	✓ Gravity_SPH	
✓ Config_magneticum							
✓ Config_muppi							
✓ Config_sh							

How to create a job?

- Essentially you could do anything you can do in a terminal on that system, e.g.:
  - Bash scripts
  - Python scripts
  - System commands
  - …


compile_config1.sh 349 B

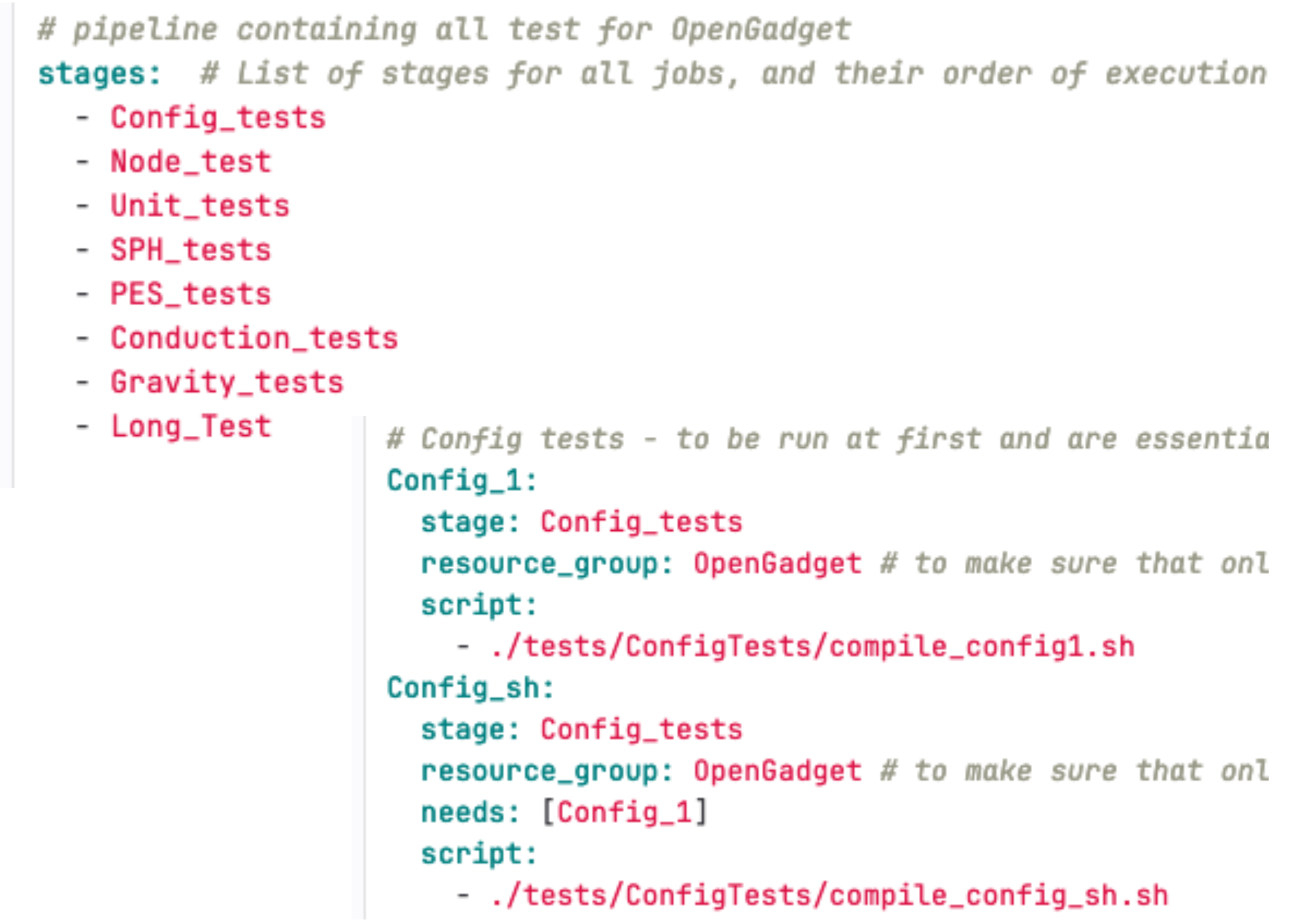
```bash
#!/bin/bash
# terminate if command returns anything else than and exit code 0
set -e

${OPENGADGET3_TEST_PREAMBLE_COMMAND}
source tests/configure_systype.sh

make clean CONFIG=tests/ConfigTests/Config1.sh
make CONFIG=tests/ConfigTests/Config1.sh -j

make clean CONFIG=tests/ConfigTests/Config1-MFM.sh
make CONFIG=tests/ConfigTests/Config1-MFM.sh -j
```


run_sedov_sph.sh 332 B

```bash
#!/bin/bash
${OPENGADGET3_TEST_PREAMBLE_COMMAND}
source tests/configure_systype.sh
rm -rf snap_???
make clean CONFIG=tests/SedovBlastwave/SEDOV-SPH.Config.sh
make CONFIG=tests/SedovBlastwave/SEDOV-SPH.Config.sh -j
$OPENGADGET3_EXE_COMMAND tests/SedovBlastwave/SEDOV-SPH.parameters.tex
python3.8 tests/SedovBlastwave/verify_sedov.py
```

Additional options:

- Stages:
  - Organize tests in categories
- Resource_group:
  - By default pipelines run concurrently – to avoid this you can define a resource group to control the concurrency.
- Needs:
  - Define requirements for job.

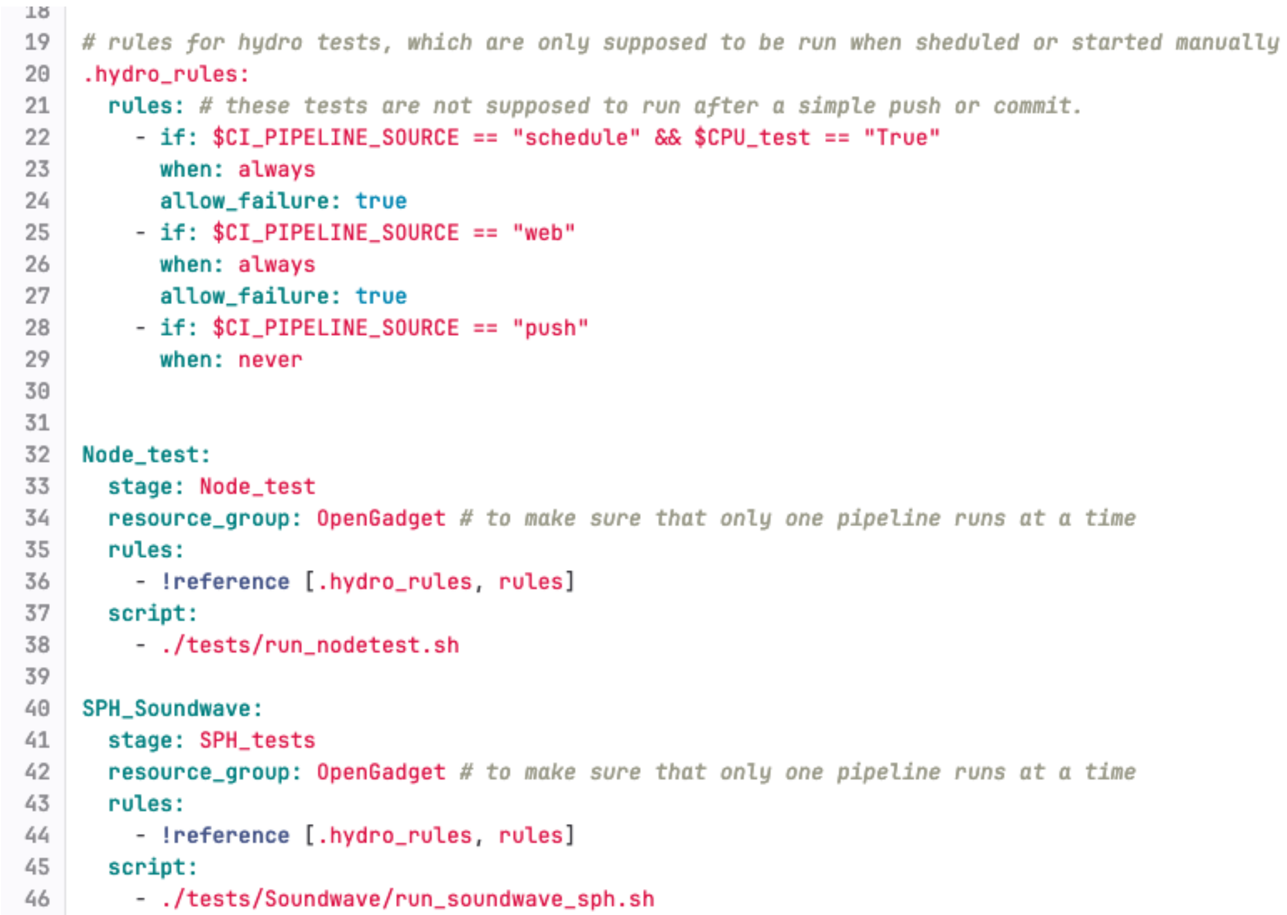


```
# pipeline containing all test for OpenGadget
stages:  # List of stages for all jobs, and their order of execution
  - Config_tests
  - Node_test
  - Unit_tests
  - SPH_tests
  - PES_tests
  - Conduction_tests
  - Gravity_tests
  - Long_Test
```

```
# Config tests - to be run at first and are essentia
Config_1:
  stage: Config_tests
  resource_group: OpenGadget # to make sure that onl
  script:
    - ./tests/ConfigTests/compile_config1.sh
Config_sh:
  stage: Config_tests
  resource_group: OpenGadget # to make sure that onl
  needs: [Config_1]
  script:
    - ./tests/ConfigTests/compile_config_sh.sh
```

Additional options:

- If statements (more details in next slide):
- When:
  - Manual: only run if triggered manually.
  - Never: don’t run the job.
  - Always: run the job regardless of earlier stages.
  - Delayed: run job after delay time.
  - on_failure: if job in previous stage failed.
  - on_success (default): if all jobs in earlier stages were succesful.
- Allow_failure:
  - Control pipeline behaviour in case of failed jobs. (default: false)

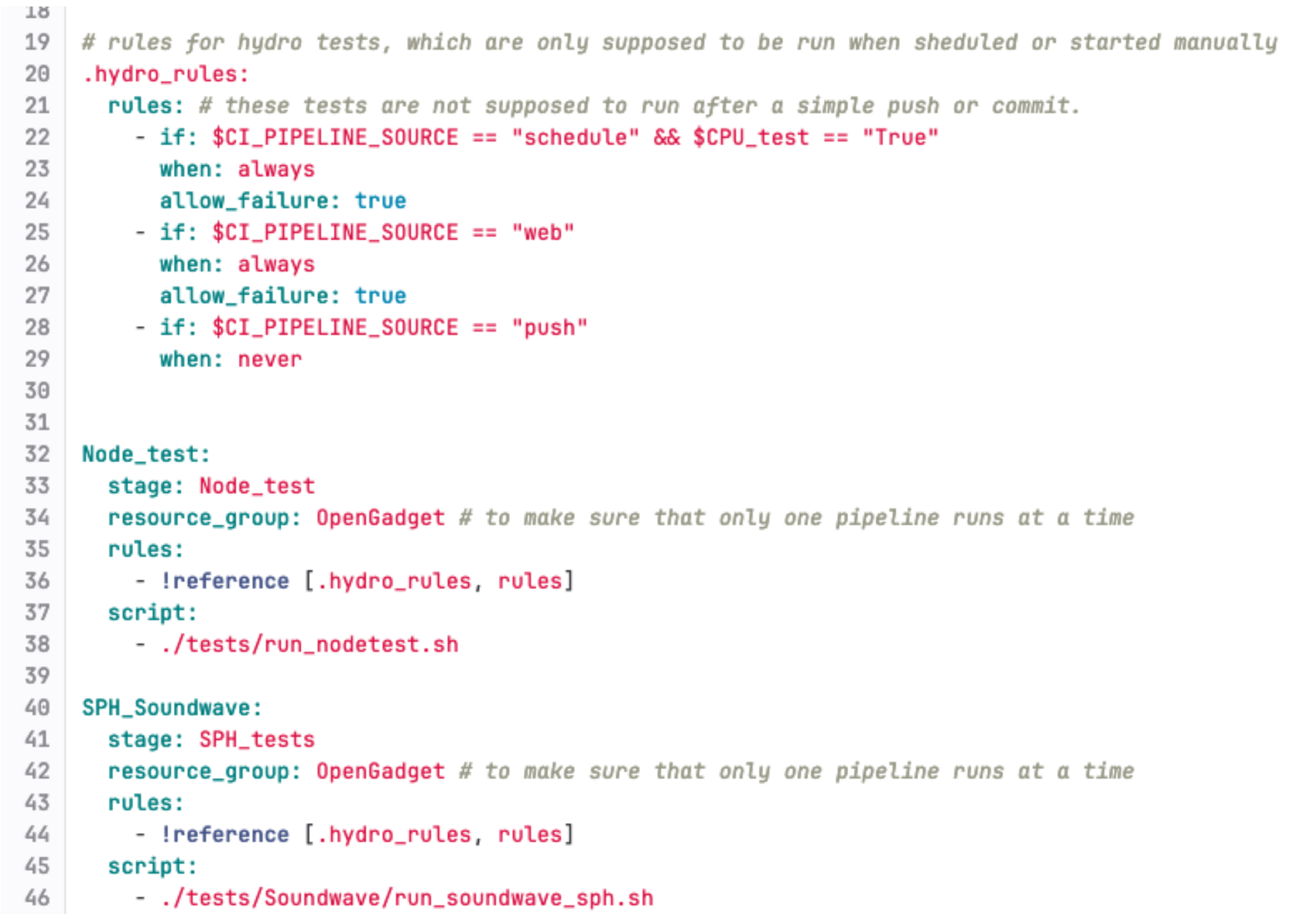


```yaml
# rules for hydro tests, which are only supposed to be run when sheduled or started manually
.hydro_rules:
  rules: # these tests are not supposed to run after a simple push or commit.
    - if: $CI_PIPELINE_SOURCE == "schedule" && $CPU_test == "True"
      when: always
      allow_failure: true
    - if: $CI_PIPELINE_SOURCE == "web"
      when: always
      allow_failure: true
    - if: $CI_PIPELINE_SOURCE == "push"
      when: never


Node_test:
  stage: Node_test
  resource_group: OpenGadget # to make sure that only one pipeline runs at a time
  rules:
    - !reference [.hydro_rules, rules]
  script:
    - ./tests/run_nodetest.sh

SPH_Soundwave:
  stage: SPH_tests
  resource_group: OpenGadget # to make sure that only one pipeline runs at a time
  rules:
    - !reference [.hydro_rules, rules]
  script:
    - ./tests/Soundwave/run_soundwave_sph.sh
```

Additional options:

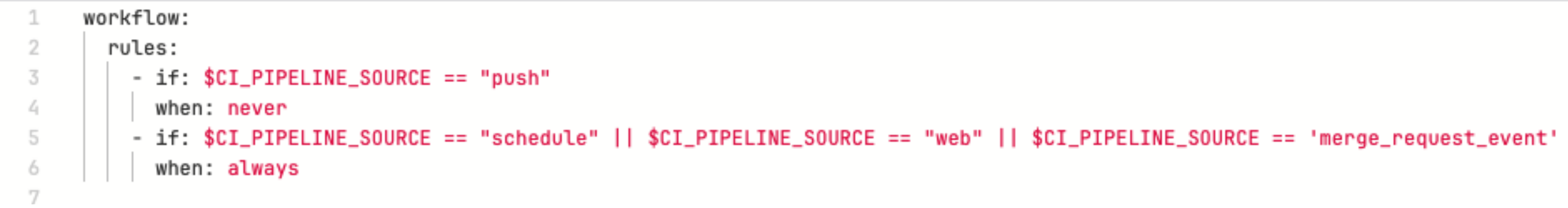
- Rules: Can be defined per job or for multiple at the same time.

- Example for rules:
  - rules:if (e.g. check if pipeline is started manually/scheduled/etc.)
  - rules:changes (e.g. check for changes in particular files)
  - rules:exists (e.g. check if specific file exists)
  - rules:allow_failure (e.g. allow test to fail)
  - rules:needs (e.g. check for requirements)
  - rules:variables (e.g. define variables)



```yaml
# rules for hydro tests, which are only supposed to be run when sheduled or started manually
.hydro_rules:
  rules: # these tests are not supposed to run after a simple push or commit.
    - if: $CI_PIPELINE_SOURCE == "schedule" && $CPU_test == "True"
      when: always
      allow_failure: true
    - if: $CI_PIPELINE_SOURCE == "web"
      when: always
      allow_failure: true
    - if: $CI_PIPELINE_SOURCE == "push"
      when: never


Node_test:
  stage: Node_test
  resource_group: OpenGadget # to make sure that only one pipeline runs at a time
  rules:
    - !reference [.hydro_rules, rules]
  script:
    - ./tests/run_nodetest.sh

SPH_Soundwave:
  stage: SPH_tests
  resource_group: OpenGadget # to make sure that only one pipeline runs at a time
  rules:
    - !reference [.hydro_rules, rules]
  script:
    - ./tests/Soundwave/run_soundwave_sph.sh
```

Additional options:



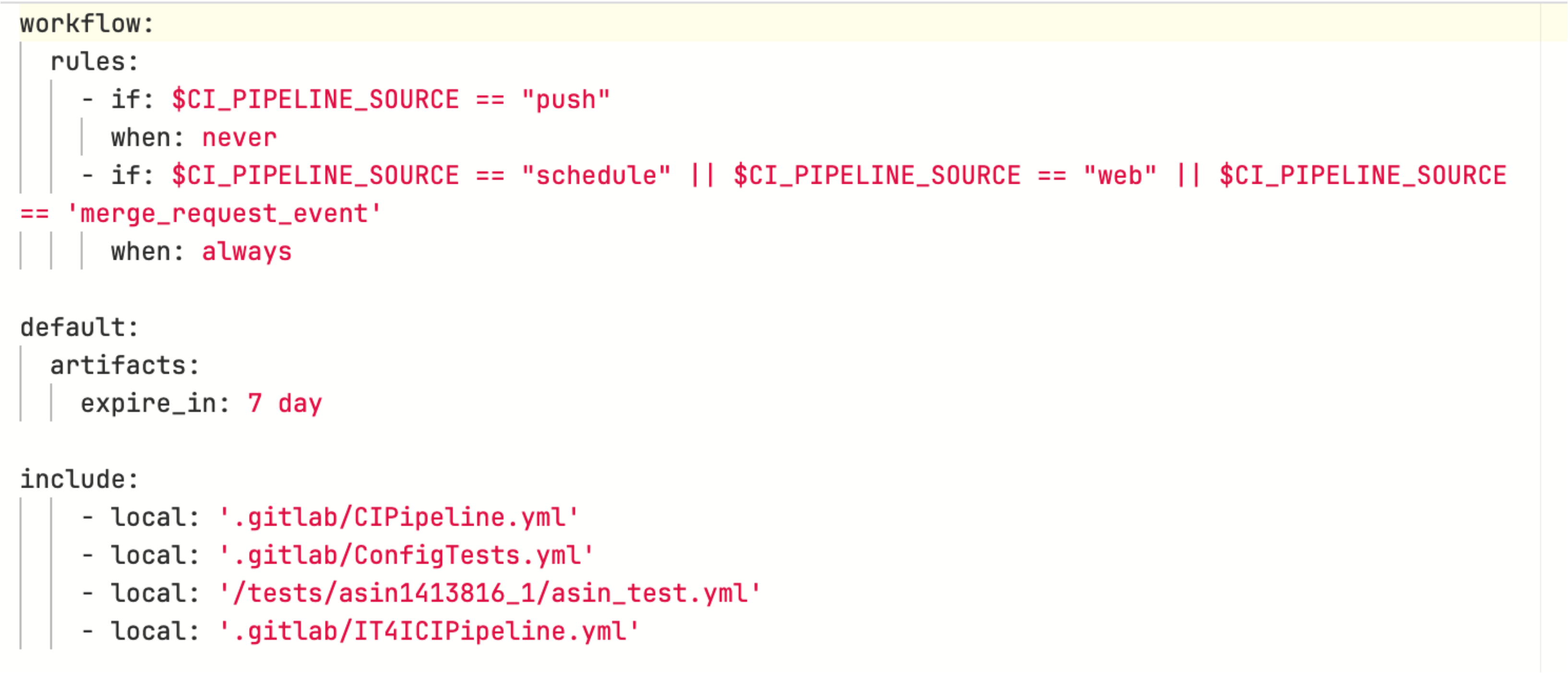
```
workflow:
  rules:
    - if: $CI_PIPELINE_SOURCE == "push"
      when: never
    - if: $CI_PIPELINE_SOURCE == "schedule" || $CI_PIPELINE_SOURCE == "web" || $CI_PIPELINE_SOURCE == 'merge_request_event'
      when: always
```

- Rules for all jobs:
  - workflow:rules (e.g. only run pipeline at particular event, on particular branch, …)

Additional options:

- Pipeline can be split over multiple files.

- These can be local, remote or templates.



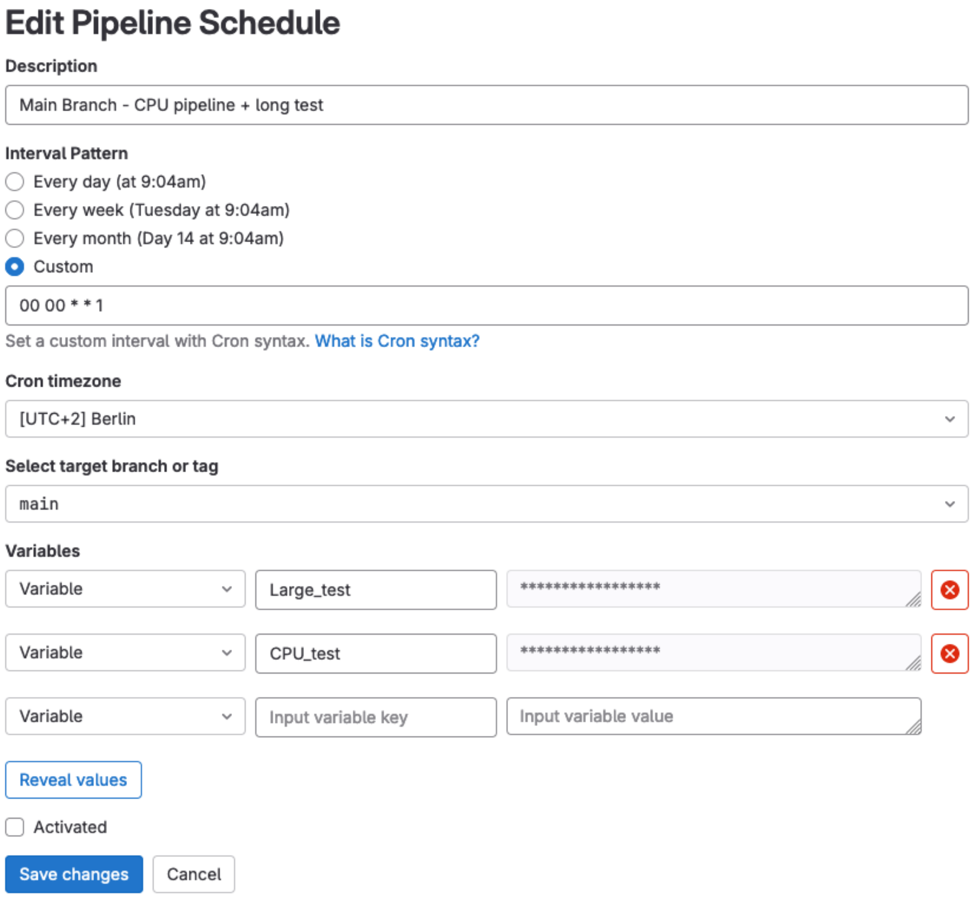
```yaml
workflow:
  rules:
    - if: $CI_PIPELINE_SOURCE == "push"
      when: never
    - if: $CI_PIPELINE_SOURCE == "schedule" || $CI_PIPELINE_SOURCE == "web" || $CI_PIPELINE_SOURCE == 'merge_request_event'
      when: always

default:
  artifacts:
    expire_in: 7 day

include:
    - local: '.gitlab/CIPipeline.yml'
    - local: '.gitlab/ConfigTests.yml'
    - local: '/tests/asin1413816_1/asin_test.yml'
    - local: '.gitlab/IT4ICIPipeline.yml'
```

Additional options:

- artifacts: define treatment of log files etc.
- default: e.g. run some script before each job.
- before_script/after_script: run before/after a job's script section (e.g. install software/set back to default)
- pages: upload test result to GitLab page, a static webpage.
- environment: Define environment on which job deploys.
- And many others… see https://docs.gitlab.com/ee/ci/yaml/ for more details.

Pipeline scheduling:

- Pipelines can be scheduled for regular advanced tests (See: Build -> Pipeline schedules)
- When: Cron syntax, e.g. every Sunday at midnight
- Where: On main branch
- Variables:
  - CPU_test: True
  - Large_test: True



Edit Pipeline Schedule

Description

Main Branch - CPU pipeline + long test

Interval Pattern

○ Every day (at 9:04am)
○ Every week (Tuesday at 9:04am)
○ Every month (Day 14 at 9:04am)
● Custom

00 00 * * 1

Set a custom interval with Cron syntax. What is Cron syntax?

Cron timezone

[UTC+2] Berlin

Select target branch or tag

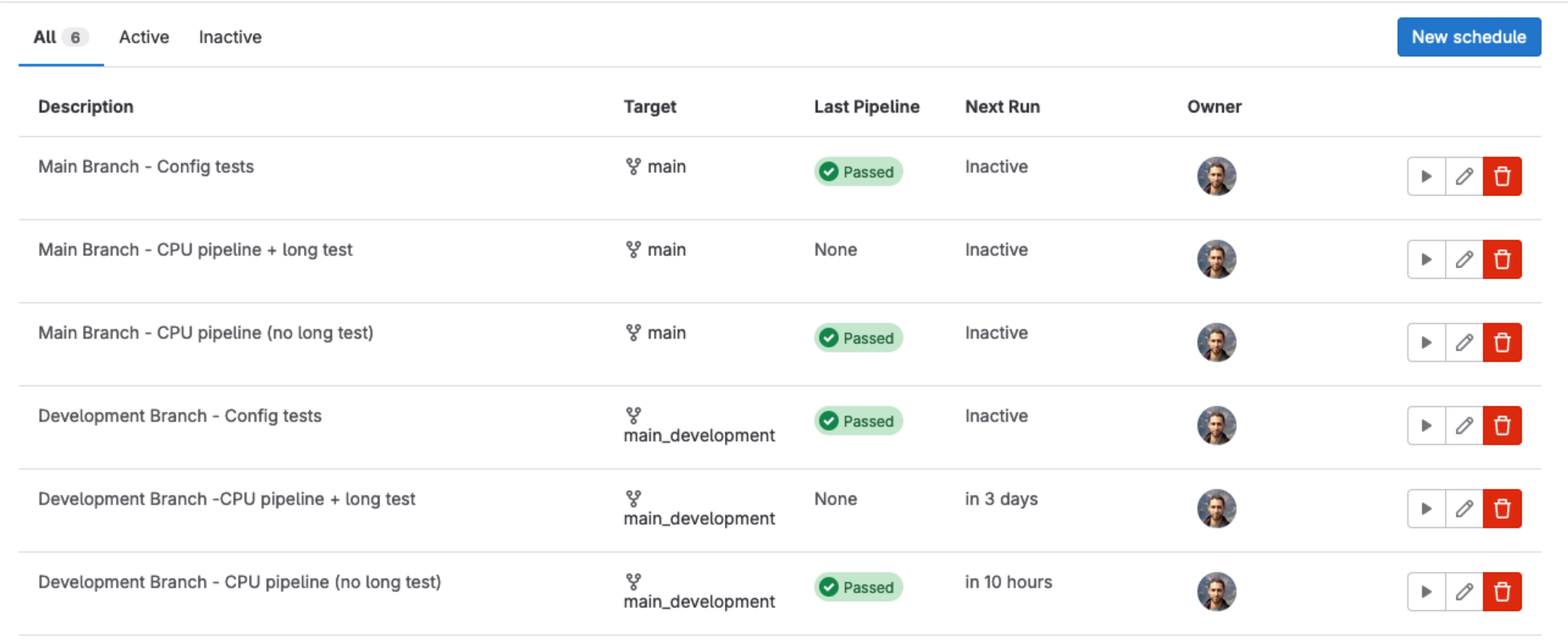
main

Variables

Variable	Large_test	*****************
Variable	CPU_test	*****************
Variable	Input variable key	Input variable value

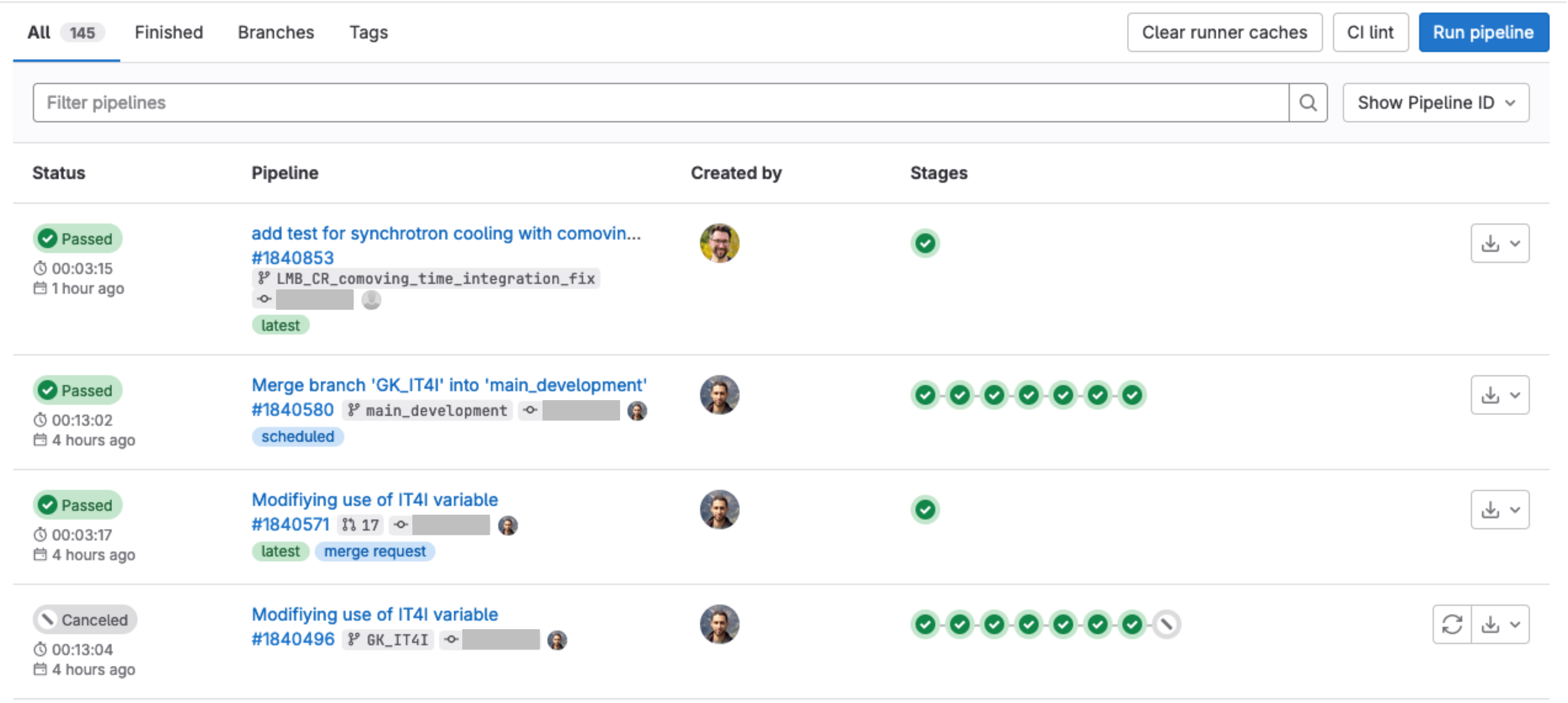
□ Activated

Pipeline scheduling:



Description	Target	Last Pipeline	Next Run	Owner
Main Branch - Config tests	main	✓ Passed	Inactive	
Main Branch - CPU pipeline + long test	main	None	Inactive	
Main Branch - CPU pipeline (no long test)	main	✓ Passed	Inactive	
Development Branch - Config tests	main_development	✓ Passed	Inactive	
Development Branch -CPU pipeline + long test	main_development	None	in 3 days	
Development Branch - CPU pipeline (no long test)	main_development	✓ Passed	in 10 hours	

Pipeline scheduling:



Status	Pipeline	Created by	Stages
Passed 00:03:15 1 hour ago	add test for synchrotron cooling with comovin... #1840853 LMB_CR_comoving_time_integration_fix [redacted] latest		✓
Passed 00:13:02 4 hours ago	Merge branch 'GK_IT4I' into 'main_development' #1840580 main_development [redacted] scheduled		✓ ✓ ✓ ✓ ✓ ✓ ✓
Passed 00:03:17 4 hours ago	Modifiying use of IT4I variable #1840571 17 [redacted] latest merge request		✓
Canceled 00:13:04 4 hours ago	Modifiying use of IT4I variable #1840496 GK_IT4I [redacted]		✓ ✓ ✓ ✓ ✓ ✓ ✓ ⊘

Sources:

- GitLab docs provide large overview and all details about runners and pipelines: https://docs.gitlab.com/

Questions?

Acknowledgement & Disclaimer

Funded by the European Union. This work has received funding from the European High Performance Computing Joint Undertaking (JU) and Belgium, Czech Republic, France, Germany, Greece, Italy, Norway, and Spain under grant agreement No 101093441.

Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European High Performance Computing Joint Undertaking (JU) and Belgium, Czech Republic, France, Germany, Greece, Italy, Norway, and Spain. Neither the European Union nor the granting authority can be held responsible for them