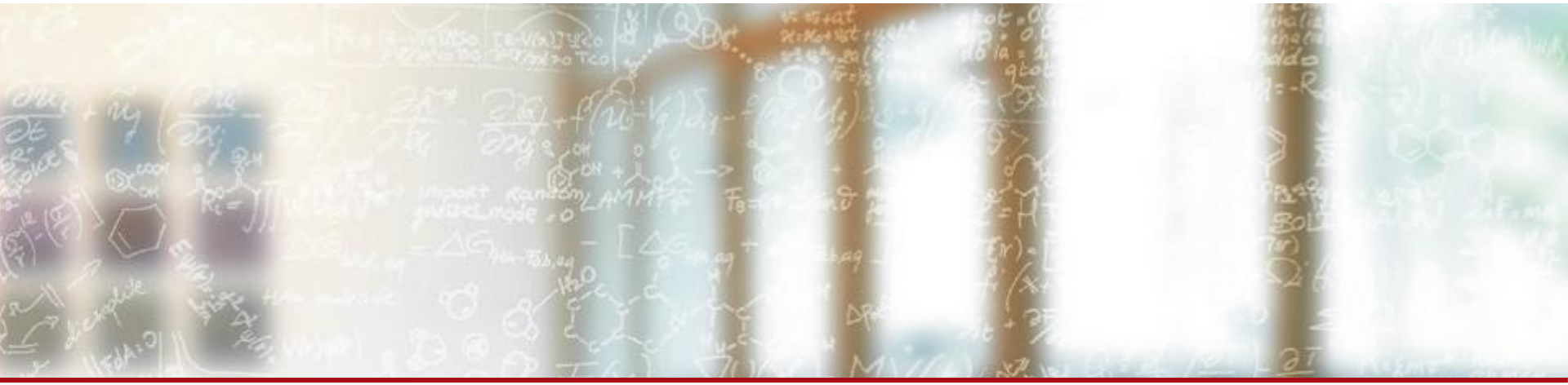


Containerized CI/CD based on GitLab runners

Andreas Fink
CSCS
September, 2022

Outline - CI/CD

1. Why?
2. How
3. Missing and possible features

Why?

- Enforcement of containers
  - Easier moving to new platforms
  - Test setup on own machine, ship same setup to HPC centre
  - Container engine is developed by CSCS, ensuring near native performance
- Enforcement of common CI/CD best practices
  - We know that a CI pipeline will (almost) always consist of a build-step and a test-step

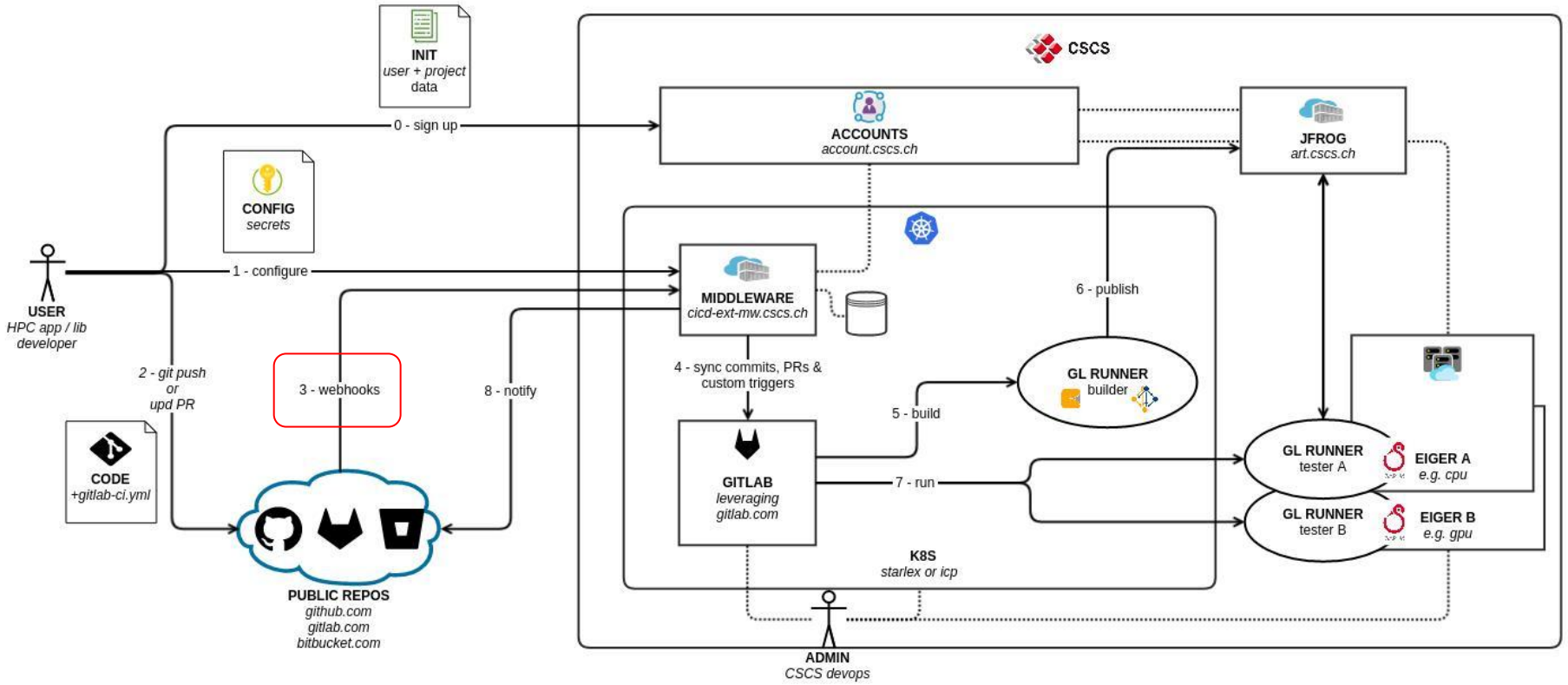
Why?

- Jenkins:
  - Allows running bare metal tests
  - Has access to Crays programming environments
  - Complicated when containerized CI is wanted
- Containerised GitLab runner setup:
  - Easy access to build containers and run them on CSCS machines
  - CI recipe is part of the repository

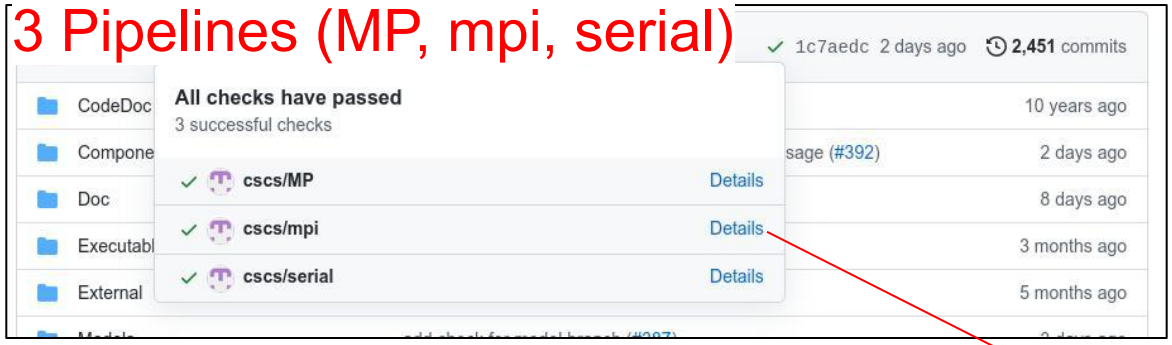
How

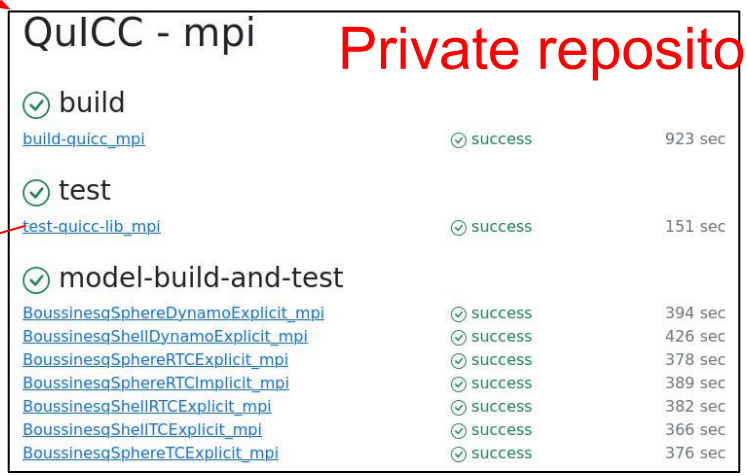
- But my project is not hosted at GitLab, how can I use your GitLab runners?
  - A middleware communicates between your repository (github, bitbucket, GitLab) and a mirror repository in GitLab
  - Middleware takes all the work to keep your repository and the mirror repository in sync (mirror repository is automatically created, no user interaction required)
  - Middleware informs your repository about build results

How - Setup


INIT
user + project data
0 - sign up
CONFIG
secrets
1 - configure
USER
HPC app / lib developer
2 - git push or upd PR
CODE
+gitlab-ci.yml
3 - webhooks
8 - notify
PUBLIC REPOS
github.com
gitlab.com
bitbucket.com
CSCS
ACCOUNTS
account.cscs.ch
JFROG
art.cscs.ch
MIDDLEWARE
cicd-ext-mw.cscs.ch
6 - publish
4 - sync commits, PRs & custom triggers
GL RUNNER
builder
5 - build
GITLAB
leveraging gitlab.com
7 - run
GL RUNNER
tester A
EIGER A
e.g. cpu
GL RUNNER
tester B
EIGER B
e.g. gpu
K8S
starlex or icp
ADMIN
CSCS devops

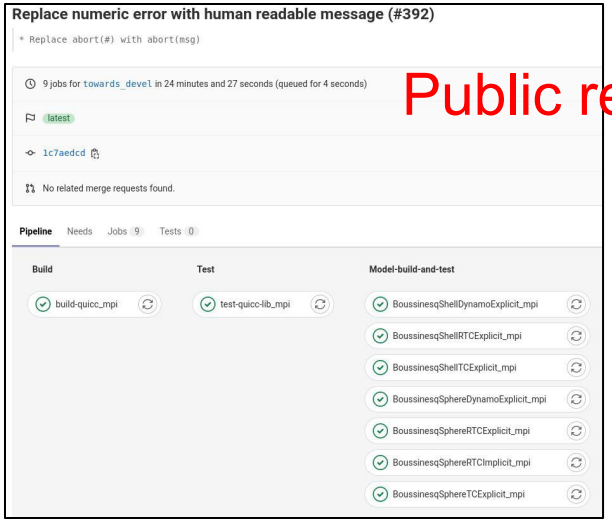
3 Pipelines (MP, mpi, serial)





Private repository

```
$ cd /QuICC.src/build
$ ctest -j $TEST_NCPU --no-tests=error --output-on-failure -R PolynomialQuadratureTests_
Test project /QuICC.src/build
    Start  4: PolynomialQuadratureTests_ChebyshevRule
1/7 Test  #4: PolynomialQuadratureTests_ChebyshevRule ..........   Passed    0.25 sec
    Start  5: PolynomialQuadratureTests_WorlandChebyshevRule
2/7 Test  #5: PolynomialQuadratureTests_WorlandChebyshevRule ...   Passed    0.23 sec
    Start  6: PolynomialQuadratureTests_WorlandSphEnergyRule
3/7 Test  #6: PolynomialQuadratureTests_WorlandSphEnergyRule ...   Passed    0.04 sec
    Start  7: PolynomialQuadratureTests_WorlandCylEnergyRule
4/7 Test  #7: PolynomialQuadratureTests_WorlandCylEnergyRule ...   Passed    0.00 sec
    Start  8: PolynomialQuadratureTests_WorlandLegendreRule
5/7 Test  #8: PolynomialQuadratureTests_WorlandLegendreRule ....   Passed    0.02 sec
    Start  9: PolynomialQuadratureTests_LegendreRule
6/7 Test  #9: PolynomialQuadratureTests_LegendreRule ...........   Passed    0.03 sec
    Start 10: PolynomialQuadratureTests_JacobiRule_ulp650
7/7 Test #10: PolynomialQuadratureTests_JacobiRule_ulp650 ......   Passed    0.25 sec

100% tests passed, 0 tests failed out of 7

Total Test time (real) =   1.26 sec
$ ctest -j $TEST_NCPU --no-tests=error --output-on-failure -R ^Jacobi
Test project /QuICC.src/build
    Start 25: JacobiBaseTest
1/2 Test #25: JacobiBaseTest ...................   Passed    0.02 sec
    Start 26: JacobiAsymptoticTest
2/2 Test #26: JacobiAsymptoticTest .............   Passed    0.01 sec

100% tests passed, 0 tests failed out of 2

Total Test time (real) =   0.05 sec
$ ctest -j $TEST_NCPU --no-tests=error --output-on-failure -R "TransformFourierTests_Complex_.*_integrator"
Test project /QuICC.src/build
      Start 222: TransformFourierTests_Complex_P_integrator
 1/10 Test #222: TransformFourierTests_Complex_P_integrator ..............   Passed    0.12 sec
      Start 223: TransformFourierTests_Complex_D1_integrator
 2/10 Test #223: TransformFourierTests_Complex_D1_integrator .............   Passed    0.01 sec
      Start 224: TransformFourierTests_Complex_D1_Neg_integrator
 3/10 Test #224: TransformFourierTests_Complex_D1_Neg_integrator .........   Passed    0.01 sec
      Start 225: TransformFourierTests_Complex_D1_P_integrator
 4/10 Test #225: TransformFourierTests_Complex_D1_P_integrator ...........   Passed    0.01 sec
      Start 226: TransformFourierTests_Complex_D2_integrator
 5/10 Test #226: TransformFourierTests_Complex_D2_integrator .............   Passed    0.01 sec
      Start 227: TransformFourierTests_Complex_Lapl2D_integrator
 6/10 Test #227: TransformFourierTests_Complex_Lapl2D_integrator .........   Passed    0.01 sec
      Start 228: TransformFourierTests_Complex_InvLapl2D_integrator
 7/10 Test #228: TransformFourierTests_Complex_InvLapl2D_integrator ......   Passed    0.01 sec
      Start 229: TransformFourierTests_Complex_Df1InvLapl2D_integrator
 8/10 Test #229: TransformFourierTests_Complex_Df1InvLapl2D_integrator ...   Passed    0.01 sec
      Start 230: TransformFourierTests_Complex_Mean_integrator
 9/10 Test #230: TransformFourierTests_Complex_Mean_integrator ...........   Passed    0.01 sec
      Start 231: TransformFourierTests_Complex_P_Clean_integrator
10/10 Test #231: TransformFourierTests_Complex_P_Clean_integrator ........   Passed    0.01 sec

100% tests passed, 0 tests failed out of 10
```



Public repository

How

1. Let your repository be registered
   a. You will receive webhook details (a webhook URL with ID, a secret)
2. Setup webhook

   (https://cicd-ext-mw.cscs.ch/ci/webhook_ci?id=MY_REPO_ID)

   a. webhooks have a secret, requests with the wrong secret are rejected
   b. repository ids are tied to a git url, i.e. you cannot reuse the same ID for another git repository webhook, the webhook events will be rejected
3. Setup CI (https://cicd-ext-mw.cscs.ch/ci/setup_ci)
   a. Credentials are username: repository id, password: webhook secret
4. Commit pipeline yml files to your repository (gitlab CI documentation is a good starting point)

How - CI and Gitlab terms

- One repository has many pipelines
- One pipeline has one entrypoint
- One pipeline has many stages
- One stage has many jobs
- Every job has a `script` tag, which contains the commands to be executed
- One pipeline creates one status report in the original repository
- Implementation detail: One pipeline is a fully mirrored repository on GitLab (because GitLab knows only of one entrypoint per repository)
- Private repos are mirrored to a private repo in gitlab

How - When is a CI pipeline triggered

- A repository has a default set of branches that are triggering on every push
- A pipeline can overwrite this set of branches
- A pull request (not from a forked repository) that is targeting one of the branches that triggers a pipeline, will also trigger pipelines
- A pull request from a forked repository is only triggering the pipeline, when the user is white-listed
- White-listed user can be set on repository level or overwritten on pipeline level

Repository ID:

123456

Repository URL:

https://github.com/finkandreas/docker-jfrog

Webhook setup details

Go to https://github.com/finkandreas/docker-jfrog/settings/hooks and add a new webhook

```
Payload URL: https://cicd-ext-mw.cscs.ch/citest/webhook_ci?id=123456
Content type: application/json
Secret: The webhook secret that you set in the global config below
Which events would you like to trigger this webhook: Send me everything
```

Global config

Repository name:

docker-jfrog devel

Private repository Make the mirror a private repository (this implies that log files cannot be viewed in gitlab directly, there is a simpler interface to view log files)

Your personal SSH deploy key is: `ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAICO7cjaM0QsTJFj7OnNVK8ekABXL1y21J+yXzfJVgg9m CSCS-CI`

Add this deploy key to your repo here

It is required to add this key such that the source code can be cloned on the target machines. It is *not* necessary to grant write acces to this deploy key.

▸ Notification token:

***

Current value is not echoed, leave it empty to keep it unchanged.

Webhook secret:

***

Current value is not echoed, leave it empty to keep it unchanged (keep in mind that the password to this setup site is the webhook secret)

▸ Default whitelisted users:

finkandreas

Comma-separated list of whitelisted users (applies to all pipelines, unless explicitly set to a different value in the pipeline config)

▸ Default CI enabled branches:

main

Comma-separated list of branch names for which CI will run on each push event (applies to all pipelines, unless explicitly set to a different value in the pipeline config)

Pipeline P100

Pipeline default

Pipeline baseimage

Submit changes Add new pipeline

Pipeline baseimage

Pipeline name:

baseimage

Keep it short (e.g. P100, A100, GPU...). It will be used as name in the status notifications (alphanumeric characters or dash allowed, no whitespace)

Pipeline YML-entrypoint:

ci/baseimage.yml

Relative path to yml file in the repository

Whitelisted users:

Comma-separated list of whitelisted users (overrides default config if not empty)

CI enabled branches:

baseimage

Comma-separated list of branch names for which CI will run on each push event (overrides default config if not empty)

⊗ Delete pipeline baseimage

Submit changes | Add new pipeline

How - Typical pipeline

- A pipeline has (typically at least) two stages
  - Build stage (`docker_jfrog` runner)
    - A new docker image is created and pushed to a registry (JFrog)
  - Test stage (`daint-container` runner)
    - The newly created image will be pulled and (MPI)-tests are being run inside the new container

How - Docker-JFrog runner

- This runner takes a Dockerfile as input and produces a new docker image as output
- Image is pushed automatically to JFrog (jfrog.svc.cscs.ch)
- Image name is static (predefined variables can be used)
- Instead of a Dockerfile the commands can be given directly too
- Hosted at

  https://gitlab.com/cscs-ci/ci-testing/webhook-ci/gitlab-runner-docker-jfrog

How - Docker-JFrog runner

Example without Dockerfile

```
tags:
  - docker_jfrog
stage: build
  image: ubuntu:22.04
  script:
    - mkdir build
    - cd build
    - cmake -DCMAKE_INSTALL_PREFIX=/opt/hello ..
    - make -j2
    - make install
variables:
  PERSIST_IMAGE_NAME: ${CSCS_REGISTRY_PATH}/my_project:1.0
```

Example with Dockerfile

```
tags:
  - docker_jfrog
stage: build
variables:
  PERSIST_IMAGE_NAME: ${CSCS_REGISTRY_PATH}/my_project:1.0
  DOCKERFILE: ci/docker/Dockerfile
```

Generated Dockerfile

```
FROM ubuntu:22.04
WORKDIR /sources
COPY . /sources
RUN buildscript.sh
```

Generated by runner and copied to the root directory of the git source directory (filename is not really buildscript.sh, to avoid overwriting files)

How - Slurm-Sarus-Runner

- This runner has a container image as input and runs executables inside the container (on N nodes with M tasks)
- All slurm environment variables are supported and alter the job setup (number of nodes, number of tasks, output, time limit, etc)
- N containers are spawned that can communicate via MPI
- Inside the container it is not possible to spawn another MPI environment, i.e. the running container is the MPI environment
- https://gitlab.com/cscs-ci/gitlab-runner-slurm-sarus

How - Slurm sarus runner

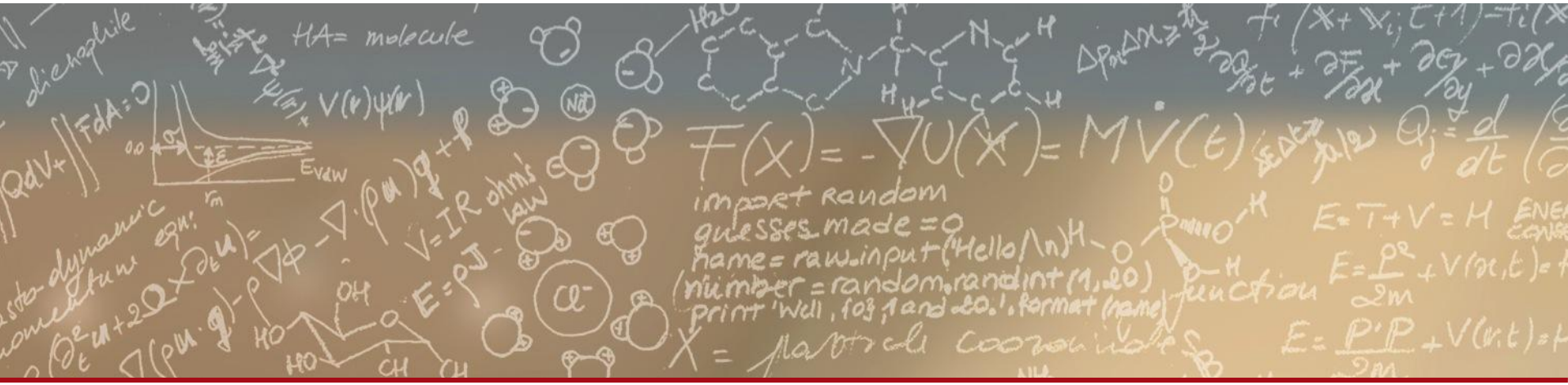
```
tags:
  - daint-container
stage: test
image: ${PERSIST_IMAGE_NAME}
script:
  - /opt/my_project/bin/my_binary
variables:
  PULL_IMAGE: 'YES'
  CSCS_REGISTRY_LOGIN: 'YES'
  SLURM_JOB_NUM_NODES: 2
  SLURM_PARTITION: normal
  SLURM_NTASKS: 2
```

How - Full yml file

```
stages: [build, test]
variables:
  PERSIST_IMAGE_NAME: $CSCS_REGISTRY_PATH/my_image:${CI_COMMIT_BRANCH}
build_job:
  tags: [docker_jfrog]
  stage: build
  variables:
    DOCKERFILE: ci/docker/Dockerfile
test_job1:
  stage: test
  image: ${PERSIST_IMAGE_NAME}
  script:
    - /opt/my_project/bin/my_binary arg_1 arg_2
    - /opt/my_project/bin/my_binary arg_A arg_B
variables:
  PULL_IMAGE: 'YES'
  CSCS_REGISTRY_LOGIN: 'YES'
  SLURM_JOB_NUM_NODES: 1
  SLURM_PARTITION: normal
  SLURM_NTASKS: 1
test_job2:
  stage: test
  image: ${PERSIST_IMAGE_NAME}
  script:
    - /opt/my_project/bin/other_binary
variables:
  PULL_IMAGE: 'YES'
  CSCS_REGISTRY_LOGIN: 'YES'
  SLURM_JOB_NUM_NODES: 2
  SLURM_PARTITION: normal
  SLURM_NTASKS: 2
```

Missing features

- Many features are still missing, it's a work in progress
- No secrets management at the moment
- Container image names are static -> no matrix pipeline setup (could be worked around by having many pipelines dynamically created with a script)
- No bare metal access
- No sarus slurm GitLab runner on Eiger (or other Alps system)
- No docker runner on Alps
  - One `docker_jfrog` runner per CPU architecture is planned



Thank you for your attention.
Time for some questions.