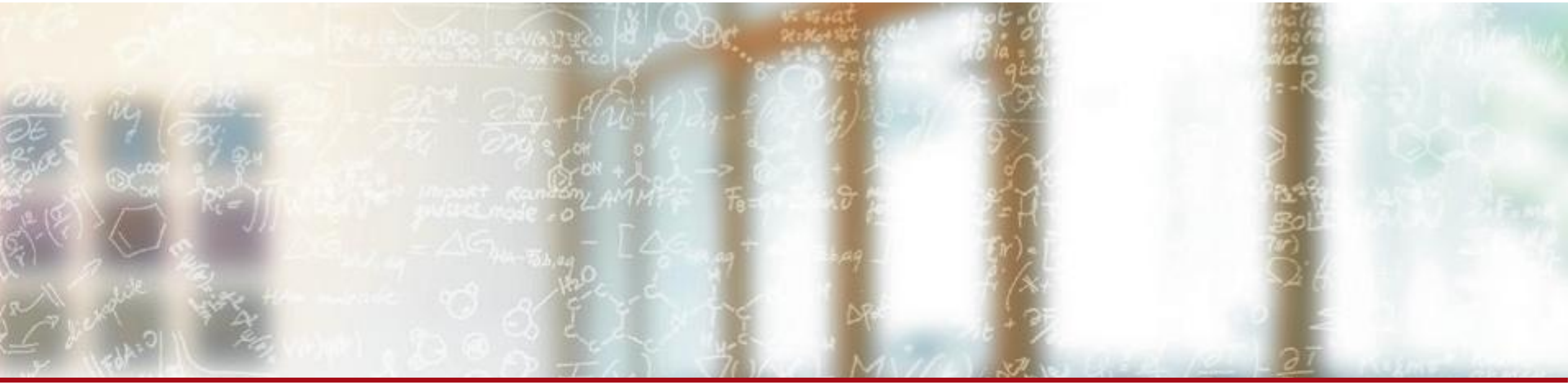
CSCS

Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich



Containerized CI/CD - Webinar

Theofilos Manitaras - CSCS
Andreas Fink - CSCS

October 4th, 2023

Why containerized?

- Track software dependencies
- Test updating software dependencies
- Test existing builds on new clusters
- (almost) Decoupled from host system
    - Test and build on own machine - promote to HPC cluster
- Sarus container runtime engine allows near native performance
- Common CI job snippets maintained centrally
- Known working base images maintained centrally
    - Including spack helpers, for guaranteed best-performance compilations per cluster

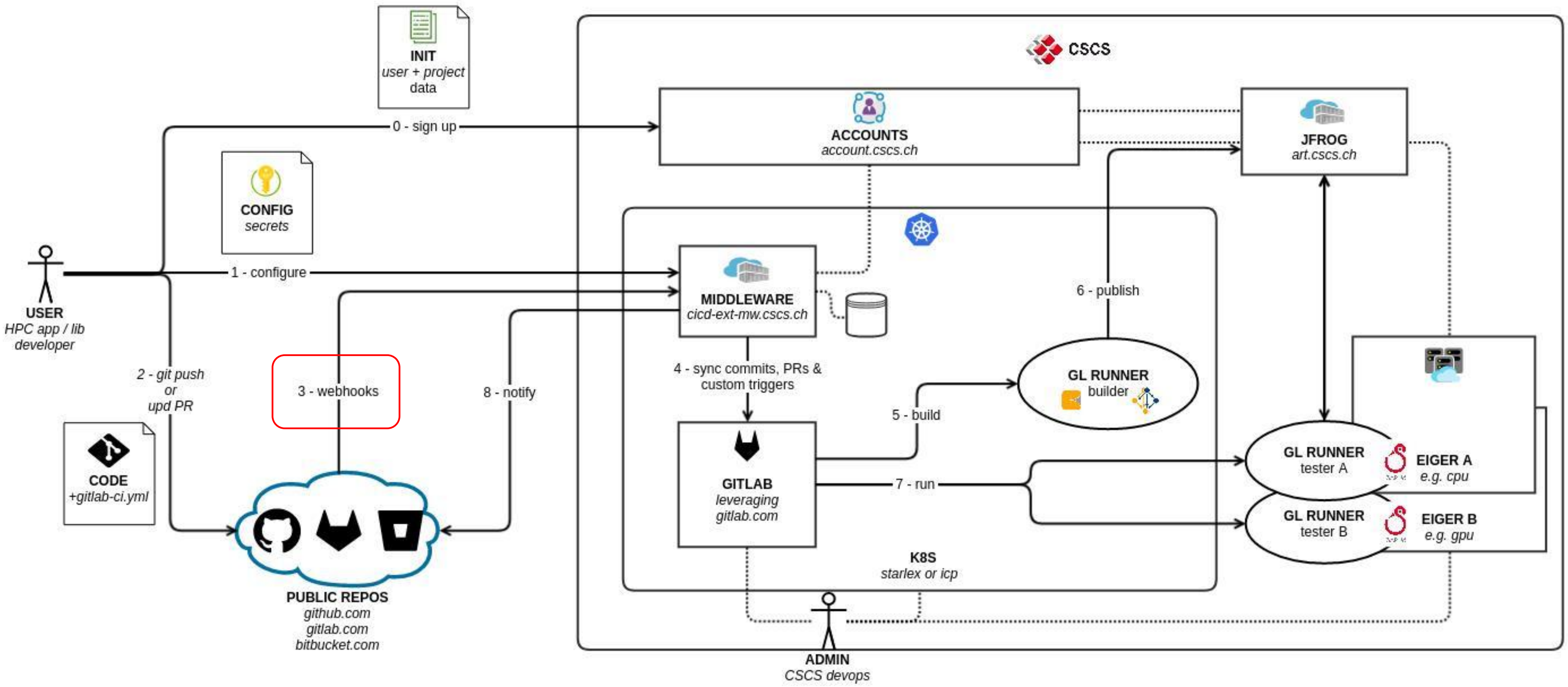
Internals

- Software can be hosted on all git providers (github.com, gitlab.com, bitbucket.org)
- Repository must setup a webhook
- Webhook sends events to a middleware orchestrator (Push-Event, Pull-Request-Event, Comment-Event, etc)
- Middleware orchestrator ensures that code is in sync between a mirror-repository and your original repository
- Mirror repository is at gitlab.com
- CI yml syntax is the same as on gitlab, i.e. the documentation of CI at gitlab is a reference
- Private repositories are mirrored privately

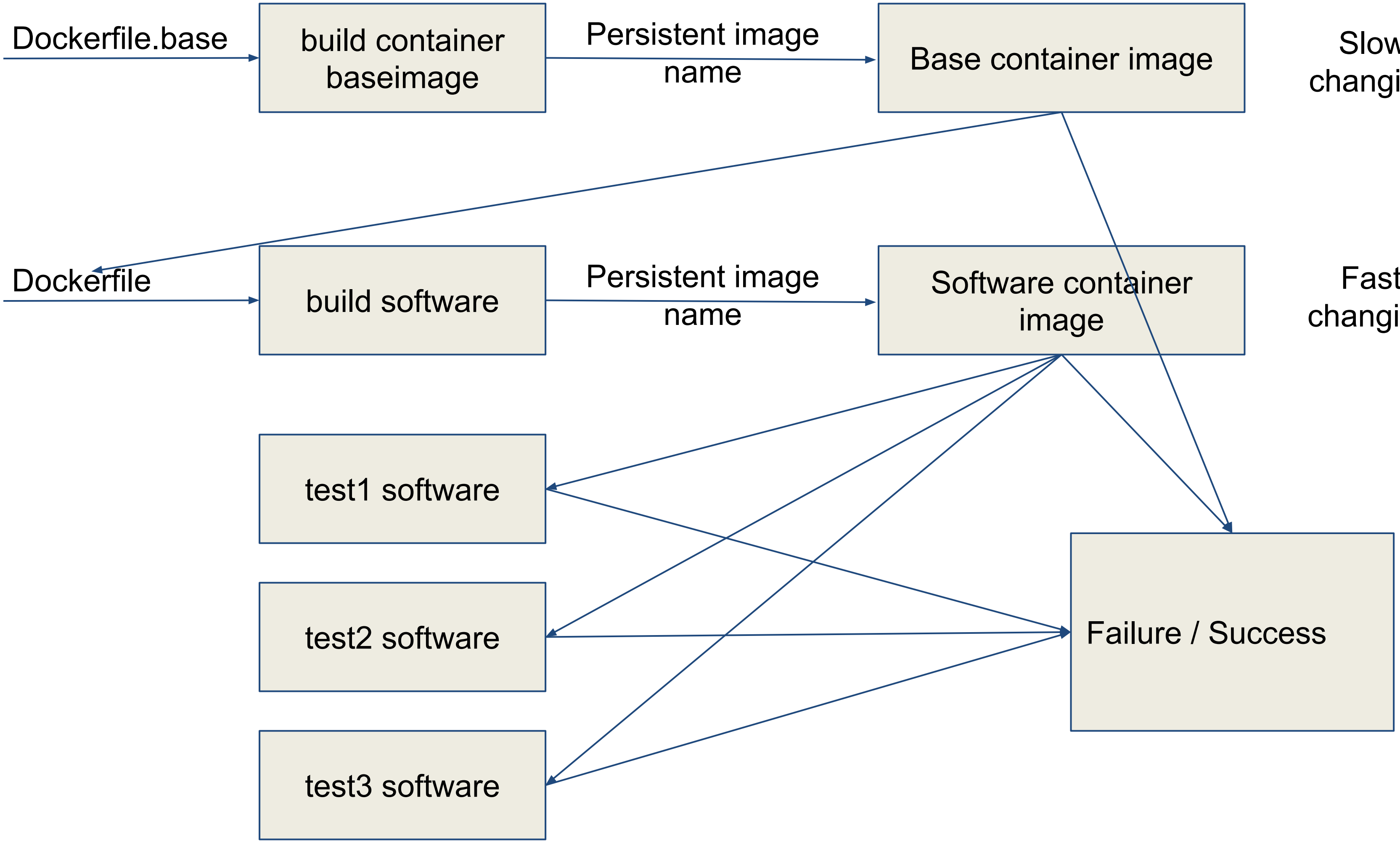
Pipeline triggers

- Push events to CI enabled branches
- PR events targeting CI enabled branches
  - automatic triggering if PR is from an in-repo branch
  - automatic triggering if PR is from a fork, but a trusted user
- Comment event “*cscs-ci run pipeline_name*”
  - Pipeline only starts if a trusted user comments on the PR
- Cron schedule for periodic builds
- API endpoint

Architecture



Typical CI setup



Traditional - Containerized CI

Traditional CI:

- Install dependencies on host system once manually.
- Build software in CI using these dependencies on the system bare-metal
- Test the software for correctness and performance

Containerized CI:

- Build a base container image
- Build a software container image based on the base container
- Test the containerized software for correctness and performance

Boilerplate CI yml

User documentation of containerized CI

```
include:
 -   remote: 'https://gitlab.com/cscs-ci/recipes/-/raw/master/templates/v2/.ci-ext.yml'

stages:
  - build
  - test
```

Job definition:

```
some job name:
  # see in the above included file for defined runner selectors
  extends: .some-runner-selector
  stage: build
  variables:
    MY_CUSTOM_VARIABLE: “some value”
    ANOTHER_VARIABLE: 42
```

Jobs in the same stage will run in parallel. Implicit barrier going from one stage to the next.

Building container images

```
build container:
  extends: .container-builder
  stage: build
  variables:
    DOCKERFILE: ci/docker/Dockerfile
    PERSIST_IMAGE_NAME: $CSCS_REGISTRY_PATH/subdir/image:$CI_COMMIT_SHORT_SHA
```

- The runner is selected by using *extends: .container-builder*
- Input argument is a Dockerfile specified in the variable *DOCKERFILE*
  - path relative to repository root directory
  - do not start with /
  - NOT relative to the path of the CI-yml file
- Output is a container image that is stored under *$PERSIST_IMAGE_NAME*
  - Must be prefixed with *$CSCS_REGISTRY_PATH*
  - Can be in any subdirectory
  - If it is stored in a subdirectory named *public*, than it can be pulled manually with sarus without credentials and tested manually

Documentation and further customization
Image retention policy

Building base images

Dockerfile

```
FROM docker.io/finkandreas/spack:0.19.2-ubuntu22.04 as builder

RUN spack-install-helper \
    daint-mc \
    cmake \
    'osu-micro-benchmarks@6.2'
```

- Start from a docker image with spack and with batteries
- Automatic build cache configured
- Built packages populate automatically the build cache
- Select correct MPI version to work with sarus' MPI hook
- Select correct micro-architecture for target system
- Select correct default spack variants (e.g. cuda_arch) for target system

Documentation and full Dockerfile example

Running containers

```
test software:
  extends: .container-runner-daint-gpu
  stage: test
  image: $CSCS_REGISTRY_PATH/subdir/image:$CI_COMMIT_SHORT_SHA
  script:
    - /path/to/binary --arg1 --arg2
  variables:
    SLURM_JOB_NUM_NODES: 2
    SLURM_LABELIO: 1
    USE_MPI: ‘YES’
```

- The runner is selected by e.g. *extends: .container-runner-daint-gpu*
- The container image is specified in *image*
- Commands to run are specified in *script*
- Slurm environment can be setup using SLURM environment variables
- *USE_MPI* instructs the runner to use the *--mpi* flag for *sarus run* which replaces the MPI inside the container with the host’s version
- Source code is not cloned by default, but can be turned on with *GIT_STRATEGY: fetch*

Documentation and further customization

Building blocks

- Separation of concerns, CSCS provides building blocks that optimize for performance for the target system.
- include .ci-ext.yml helper in your YML file (https://gitlab.com/cscs-ci/recipes/-/raw/master/templates/v2/.ci-ext.yml)
- Use *FROM finkandreas/spack:0.19.2-ubuntu22.04* in your Dockerfile to build base containers
  - it contains the helper script *spack-install-helper*
  - *spack-install-helper TARGET_SYSTEM \*
    cmake \
    ‘trilinos@13.4.0+amesos2+belos~epetra cxxstd=17’
  - See also documentation and other available tags at Dockerhub
- Use *.container-builder-dynamic-name*, defined in .ci-ext.yml
- Have a look at the example projects linked on the ci-doc

Future work

- Building blocks for future Alps hardware (Grace-Hopper)
- Secrets/Variables management
- Integrate CSCS-Single-Sign-On
- Firecrest integration
    - Allow dispatching jobs on the compute node as baremetal job
    - Allow dispatching jobs to different compute centers (e.g. LUMI will be a future target)
    - Better integration with Reframe