



CSCS

Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich



CI/CD on Alps

Andreas Fink - CSCS

andreas.fink@cscs.ch

April 2nd, 2025

Example of a CI file

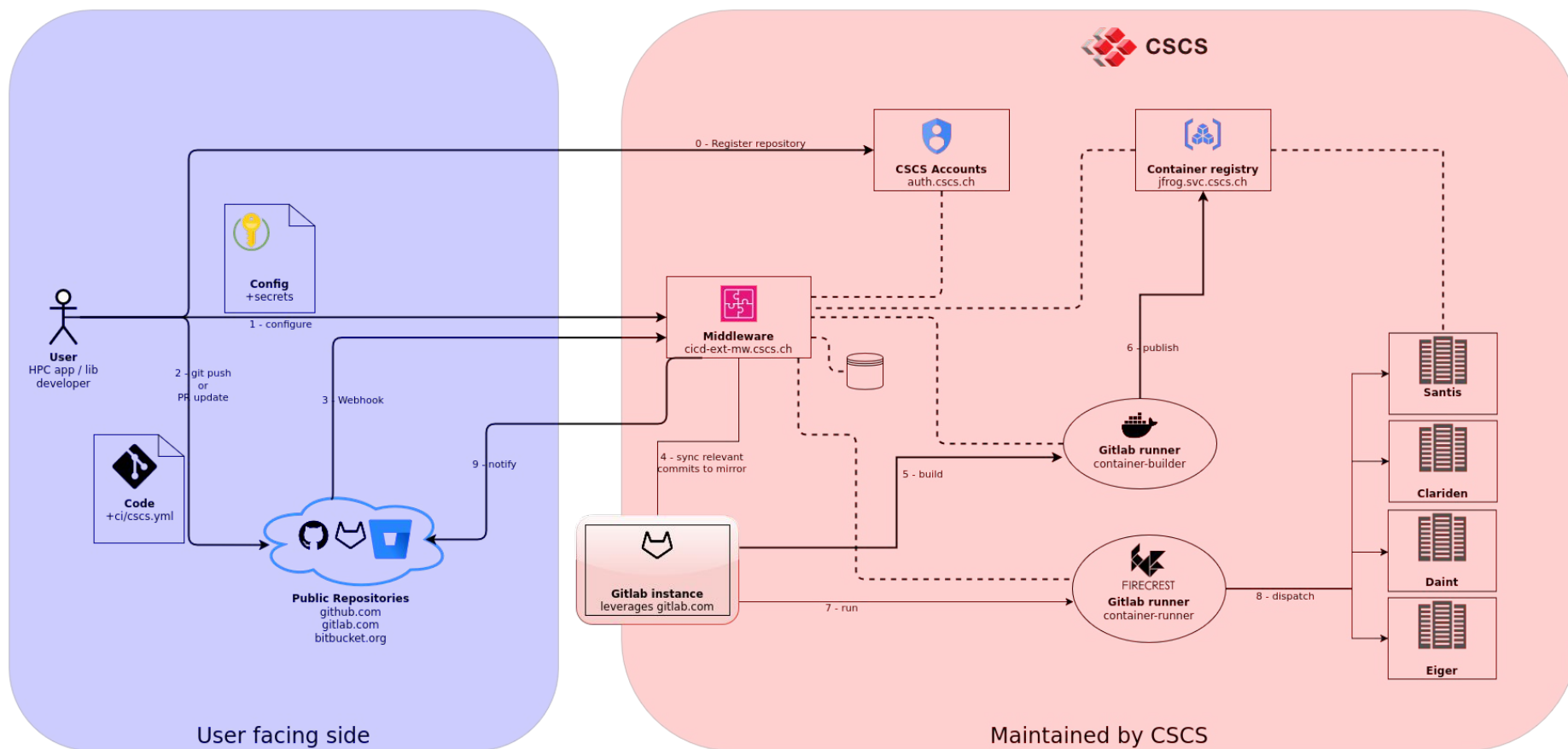
```
include:
  - remote: 'https://gitlab.com/cscs-ci/recipes/-/raw/master/templates/v2/.ci-ext.yml'

stages:
  - build
  - test

build job:
  extends: .container-builder-cscs-gh200
  stage: build
  variables:
    DOCKERFILE: ci/docker/Dockerfile
    PERSIST_IMAGE_NAME: $CSCS_REGISTRY_PATH/x86_64/my_image:$CI_COMMIT_SHORT_SHA

test feature A:
  extends: .container-runner-cscs-gh200
  image: $CSCS_REGISTRY_PATH/x86_64/my_image:$CI_COMMIT_SHORT_SHA
  stage: test
  script:
    - cd /opt/installdir
    - my_application
  variables:
    SLURM_JOB_NUM_NODES: 4
    SLURM_NTASKS: 16
```

CI/CD architecture



Your source code will be mirrored to gitlab.com

Visibility in gitlab.com will be private, if the source repository is private

CI/CD at CSCS

- Useful links:
 - CSCS Documentation: <https://confluence.cscs.ch/x/UAXJMw>
 - Gitlab documentation: <https://docs.gitlab.com/ee/ci/yaml/>
 - CI variables:
https://docs.gitlab.com/ee/ci/variables/predefined_variables.html
 - User's slack space: <https://cscs-users.slack.com> (#ci-cd channel)
- Getting access for CI/CD at CSCS:
 - Open a Service Desk ticket to register your git repository
 - Follow the rest of the steps described in above “CSCS Documentation” (section “Enable CI for your project”)

Live Demo

- Firecrest credentials creation at <https://developer.cscs.ch>
- Setup CI page at <https://cicd-ext-mw.cscs.ch>
 - Webhook setup
 - Admin config
 - Global config
 - Pipeline config

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

Runner modes

- container-builder
- container-runner
- uenv-builder
- uenv-runner
- baremetal-runner
- container-runner-lightweight
- f7t-controller
- reframe-runner

Runners - .container-builder

```
some job name:
  extends: .container-builder-cscs-zen2
  variables:
    DOCKERFILE: ci/docker/Dockerfile
    PERSIST_IMAGE_NAME: $CSCS_REGISTRY_PATH/x86_64/my_image:$CI_COMMIT_SHORT_SHA
```

Variables:

- *DOCKERFILE*: relative path to Dockerfile (mandatory)
- *PERSIST_IMAGE_NAME*: Where the container image will be stored
- *CSCS_BUILD_IN_MEMORY*: Put the whole build process in memory
- *DOCKER_BUILD_ARGS*: equivalent to `--build-arg` for `docker build`
- *CSCS_REBUILD_POLICY*: Rebuild *always* or *if-not-exists*
- *SECONDARY_REGISTRY*: Push image to a second target path
- *SECONDARY_REGISTRY_USERNAME*: Username for second target path
- *SECONDARY_REGISTRY_PASSWORD*: Password for second target path
- *CUSTOM_REGISTRY_USERNAME*: Username if not pushing to CSCS registry
- *CUSTOM_REGISTRY_PASSWORD*: Password if not pushing to CSCS registry

Runners - .container-runner

```
job1:
  extends: .container-runner-daint-gh200
  image: $CSCS_REGISTRY_PATH/aarch64/my_image:$CI_COMMIT_SHORT_SHA
  script:
    - /usr/bin/my_application /data/some_input.xml
  variables:
    CSCS_ADDITIONAL_MOUNTS: '["/capstor/scratch/cscs/<my_username>/data:/data"]'
```

Variables:

- *GIT_STRATEGY*: Clone source code if needed
- *CSCS_CUDA_MPS*: Allow multiple MPI ranks per GPU
- *CSCS_ADDITIONAL_MOUNTS*: Mount host paths inside the container runtime

Runners - .uenv-builder

```
uenv-builder-gh200:  
  extends: .uenv-builder-daint-gh200  
  variables:  
    UENV_RECIPE: ci/uenv-recipes/gh200  
    UENV_NAME: my-uenv  
    UENV_VERSION: 25.04  
    SLURM_TIMELIMIT: '01:30:00'
```

Variables:

- *UENV_RECIPE*: directory to UENV recipe (mandatory)
- *UENV_NAME*: The name of the UENV (mandatory)
- *UENV_VERSION*: Put the whole build process in memory
- *UENV_TAG*: Explicit tag
 - Do not set for auto-versioning based on the contents in *UENV_RECIPE*

Runners - .uenv-runner

```
job1:
  extends: .uenv-runner-daint-gh200
  image: my_uenv/25.04:$CI_PIPELINE_ID
  script:
    - mkdir build && cd build
    - cmake -DCMAKE_BUILD_TYPE=Release ..
    - make -j200
    - ./my_application ../data/some_input.yaml
  variables:
    WITH_UENV_VIEW: default
```

Variables:

- *WITH_UENV_VIEW*: load the UENV view
- *CSCS_CUDA_MPS*: Allow multiple MPI ranks per GPU

Runners - .baremetal-runner

```
job:
  extends: .baremetal-runner-daint-gh200
  script:
    - hostname
    - srun -n2 -N2 my_application
    - srun -n4 -N2 my_other_application
  variables:
    SLURM_JOB_NUM_NODES: 2
    SLURM_TIMELIMIT: '01:00:00'
```

- Very similar what you are used to from *sbatch*
- No convenience in any form, just baremetal access to the compute node
- Mostly equivalent to this sbatch script

```
#!/bin/bash -l
#SBATCH --nodes 2
#SBATCH --time=01:00:00

hostname
srun -n2 -N2 my_application
srun -n4 -N2 my_other_application
```

Runners - .container-runner-lightweight

```
job:
  extends: .container-runner-lightweight-zen2
  image: docker.io/python:3.11
  script:
    - ci/pipeline/generate_pipeline.py > dynamic_pipeline.yaml
  artifacts:
    paths:
      - dynamic_pipeline.yaml
```

Variables:

- *KUBERNETES_CPU_REQUEST*: Request that many CPUs
- *KUBERNETES_CPU_LIMIT*: Limit job to that many CPUs
- *KUBERNETES_MEMORY_REQUEST*: Requested amount of memory
- *KUBERNETES_MEMORY_LIMIT*: Limit amount of memory

Notes:

- LIMIT must always be larger than REQUEST
- Runner allows 1-4 CPUs and less or equal than 4Gi memory
- Maximum runtime 60 minutes
- Works only with public images

Runners - .f7t-controller

```
job:
  extends: .f7t-controller
  script:
    - CLUSTER=eiger
    - SUBMISSION="$(firecrest submit --system $CLUSTER --account
$CSCS_CI_DEFAULT_SLURM_ACCOUNT script.sh)"
    - JOBID=$(echo "$SUBMISSION" | grep "jobid" | sed -e
's/.*jobid[^0-9]*\([0-9]\+\)/\1/')
    - |
      while firecrest poll-active --raw --system $CLUSTER | grep $JOBID ; do
        echo "job is still in queue/running"
        sleep 30
      done
```

Notes:

- You are running on a lightweight machine that has the firecrest client and pyfirecrest installed
- *firecrest* client just works
- code using pyfirecrest can use the environment variables *AUTH_TOKEN_URL*, *FIRECREST_URL*, *FIRECREST_CLIENT_ID* and *FIRECREST_CLIENT_SECRET*

Runners - .reframe-runner

```
job:
  before_script:
    - git clone -b alps https://github.com/eth-cscs/cscs-reframe-tests
    - pip install -r cscs-reframe-tests/config/utilities/requirements.txt
    - sed -i -e "s/account=cstaff/account=$CSCS_CI_DEFAULT_SLURM_ACCOUNT/"
      cscs-reframe-tests/config/systems-firecrest/eiger.py
  variables:
    FIRECREST_SYSTEM: 'eiger'
    FIRECREST_BASEDIR: /capstor/scratch/cscs/jenkssl/reframe-runner
    RFM_VERSION: '4.6.2'
    RFM_FIRECREST: '1'
    RFM_CONFIG: cscs-reframe-tests/config/cscs.py
    RFM_CHECKPATH: cscs-reframe-tests/checks/microbenchmarks/mpi/halo_exchange
```

Notes:

- You are running on a machine that has the firecrest client and the pyfirecrest library installed.
- The requested ReFrame version is available
- ReFrame does not have direct access to the filesystem of the cluster so the stage directory will need to be kept in sync through FirecREST
- Further information at <https://github.com/eth-cscs/cscs-reframe-tests/blob/alps/config/systems-firecrest/README.md>

Parametrizing jobs - extends

```
.build_helper:
  variables:
    DOCKERFILE: ci/docker/Dockerfile
    PERSIST_IMAGE_NAME: $CSCS_REGISTRY_PATH/$ARCH/my_image:$CI_COMMIT_SHORT_SHA
build x86_64:
  extends: [.container-builder-cscs-zen2, .build_helper]
build aarch64:
  extends: [.container-builder-cscs-gh200, .build_helper]
```

- Helper blocks should start with a DOT, otherwise they are treated by gitlab as a full job which will be added to the pipeline
- Common things are in the helper blocks, variations are the instantiations

Parametrizing jobs - parallel:matrix

```
build:
  stage: build
  extends: .container-runner-lightweight-zen2
  image: docker.io/ubuntu:24.04
  script:
    - echo $MY_VAR1 $MY_VAR2
  parallel:
    matrix:
      - MY_VAR1: [some_value, other_value]
        MY_VAR2:
          - var2_value
          - var2_other_value
          - var2_third_value
```

- Create for each combination in parallel:matrix a job
- Some restrictions apply for the variable values due to gitlab
 - All restrictions are described at <https://docs.gitlab.com/ee/ci/yaml/#parallelmatrix>
- It is not possible to use the keyword *needs* usefully with a matrix

Parametrizing jobs - dynamic child pipelines

```
stages: [generate, run]

gen_pipeline:
  stage: generate
  extends: .container-runner-lightweight-zen2
  image: docker.io/python:3.11
  script:
    - ci/pipeline/generate_pipeline.py > dynamic_pipeline.yaml
  artifacts:
    paths:
      - dynamic_pipeline.yaml

trigger child pipeline:
  stage: run
  trigger:
    include:
      - artifact: dynamic_pipeline.yaml
        job: gen_pipeline
        forward:
          pipeline_variables: true
```

- The generated file is itself a fully valid pipeline YAML file
- Allows to generate pipelines where the number of jobs is dynamic

Dependency management

- The goal is to have fast turn around from pushing new code and CI results
- Only rebuild what has actually changed
- Containerized setup should rebuild base images if and only if
 - Dockerfile changed
 - build arguments changed
 - files *COPY/ADD*-ed into the base image have changed
- UENV-ized setup should rebuild if and only if
 - Recipe changed

Rebuilding images only when needed

```
build image:
  extends: .container-builder-cscs-zen2
  stage: build
  before_script:
    - export TAG=`cat ci/docker/Dockerfile | sha256sum - | head -c 16`
    - export PERSIST_IMAGE_NAME=$CSCS_REGISTRY_PATH/img:$TAG
    - echo "TAG=$TAG" > build.env
  artifacts:
    reports:
      dotenv: build.env
  variables:
    DOCKERFILE: ci/docker/Dockerfile
    CSCS_REBUILD_POLICY: "if-not-exists"
```

- Use hash of Dockerfile as a tag
- Save the tag as an artifact to use it in other stages / jobs
- Set `CSCS_REBUILD_POLICY: "if-not-exists"`
 - The runner will check if `$PERSIST_IMAGE_NAME` exists already

Rebuilding images only when needed

```
build image:  
  extends: [.container-builder-cscs-zen2, .dynamic-image-name]  
  stage: build  
  variables:  
    DOCKERFILE: ci/docker/Dockerfile  
    PERSIST_IMAGE_NAME: $CSCS_REGISTRY_PATH/img  
    WATCH_FILECHANGES: "ci/docker/Dockerfile"  
    CSCS_REBUILD_POLICY: "if-not-exists"
```

- Extend from .dynamic-image-name
 - Computes hash of files listed in \$WATCH_FILECHANGES
 - Sets \$BASE_IMAGE in the artifact
- Set PERSIST_IMAGE_NAME without tag
- Set CSCS_REBUILD_POLICY: "if-not-exists"

Rebuilding with build arguments

```
build image:  
  extends: [.container-builder-cscs-zen2, .dynamic-image-name]  
  stage: build  
  variables:  
    DOCKERFILE: ci/docker/Dockerfile  
    DOCKER_BUILD_ARGS: '[ "PYTHON_VER=$PYTHON_VER" ]'  
    PYTHON_VER: "3.12"  
    PERSIST_IMAGE_NAME: $CSCS_REGISTRY_PATH/python-$PYTHON_VER/img  
    WATCH_FILECHANGES: "ci/docker/Dockerfile"  
    CSCS_REBUILD_POLICY: "if-not-exists"
```

- It is easiest to make the build arguments part of the registry path
 - If the argument changes, the path changes and a rebuild is triggered

Building multiarch images

```
build multiarch image:
  stage: build_multiarch
  extends: .make-multiarch-image
  variables:
    PERSIST_IMAGE_NAME_X86_64: "<input x86_64 image>"
    PERSIST_IMAGE_NAME_AARCH64: "<input aarch64 image>"
    PERSIST_IMAGE_NAME: "<output multiarch image>"
```

- Use *.make-multiarch-image* runner template
- Takes two existing images
 - *PERSIST_IMAGE_NAME_X86_64*
 - *PERSIST_IMAGE_NAME_AARCH64*
- Creates a multiarch image
 - *PERSIST_IMAGE_NAME*

Containerized setup - Spack base images

- Available at
 - \$CSCS_REGISTRY/docker-ci-ext/base-containers/public
 - ghcr.io/eth-cscs/docker-ci-ext/base-containers/spack-base
- Preinstalled Spack with reasonable defaults and helper scripts

Multi-stage docker build:

1. Install dependencies with Spack install helper script

```
$ spack-install-helper --target <target arch> <list of specs>
```

```
$ spack-install-helper --target alps-gh200 "git" "cmake" "valgrind" "python"
```

2. Copy only installed software without Spack and other build dependencies

Spack base image

```
FROM
ghcr.io/eth-cscs/docker-ci-ext/base-containers/spack-build:spack0.23.0-ubuntu22.
04-cuda12.4.1 as builder

RUN spack-install-helper --target alps-gh200 \
    "git" "cmake" "valgrind" "python"

# end of builder container, now we are ready to copy necessary files

# copy only relevant parts to the final container
FROM
ghcr.io/eth-cscs/docker-ci-ext/base-containers/spack-runtime:ubuntu22.04-cuda12.
4.1

# it is important to keep the paths, otherwise your installation is broken
# all these paths are created with the above `spack-install-helper` invocation
COPY --from=builder /opt/spack-environment /opt/spack-environment
COPY --from=builder /opt/software /opt/software
COPY --from=builder /opt/._view /opt/._view
COPY --from=builder /etc/profile.d/z10_spack_environment.sh
/etc/profile.d/z10_spack_environment.sh

# Some boilerplate to get all paths correctly - fix_spack_install is part of the
base image
# and makes sure that all important things are being correctly setup
RUN fix_spack_install
```

Use images built in CI pipeline

- Images built in the CI pipeline are pushed to CSCS JFrog registry
- Main use case: CI pipelines
- If the image is in directory **/public/** it can be accessed from CSCS network without authentication
 - You can find the URL at the end of the build job log
 - Not recommended to use the images on JFrog directly
- Additionally you can push the images to your registry by setting these variables in the `.container-builder-*` job
 - `SECONDARY_REGISTRY`
 - `SECONDARY_REGISTRY_USERNAME`
 - `SECONDARY_REGISTRY_PASSWORD`
- Be careful if you push images to directory **/public/**, it can be accessed by everyone who knows the URL
 - Make sure such images do not contain any information that should be kept private, e.g. secrets, tokens, ... (side note - never store secrets, tokens, etc in the image, even if private)

Full containerized CI example

- Split building images into two stages
 - Dependencies
 - Base images with preinstalled Spack
 - Rebuild only when dependencies change
 - App image
 - Copy source code from git repository
 - Build the app
 - Rebuild every time
- Build multiarch image to support different architectures
- Run tests

Example: Four stages

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

Example: base image

```
build_base_image_x86_64:
  extends: [.container-builder-cscs-zen2, .dynamic-image-name]
  stage: build_base
  variables:
    DOCKERFILE: ci/docker/Dockerfile.base
    WATCH_FILECHANGES: 'ci/docker/Dockerfile.base'
    PERSIST_IMAGE_NAME: $CSCS_REGISTRY_PATH/baseimg-x86_64
    CSCS_REBUILD_POLICY: "if-not-exists"
    DOCKER_BUILD_ARGS:
'["IMG_BUILD=ghcr.io/eth-cscs/docker-ci-ext/base-containers/spack-base:spack0.21
.0-ubuntu22.04-cpu",
"IMG_RUNTIME=ghcr.io/eth-cscs/docker-ci-ext/base-containers/spack-helper:ubuntu2
2.04-cpu", "TARGET=alps-zen2"]'
```

Example: base image

```
build_base_image_aarch64:
  extends: [.container-builder-cscs-gh200, .dynamic-image-name]
  stage: build_base
  variables:
    DOCKERFILE: ci/docker/Dockerfile.base
    WATCH_FILECHANGES: 'ci/docker/Dockerfile.base'
    PERSIST_IMAGE_NAME: $CSCS_REGISTRY_PATH/baseimg-aarch64
    CSCS_REBUILD_POLICY: "if-not-exists"
    DOCKER_BUILD_ARGS:
'["IMG_BUILD=ghcr.io/eth-cscs/docker-ci-ext/base-containers/spack-base:spack0.21
.0-ubuntu22.04-cuda12.4.1",
"IMG_RUNTIME=ghcr.io/eth-cscs/docker-ci-ext/base-containers/spack-helper:ubuntu2
2.04-cuda12.4.1", "TARGET=alps-gh200"]'
```

Example: app image

```
build_app_image_x86_64:
  extends: .container-builder-cscs-zen2
  stage: build_app
  needs:
    - job: build_base_image_x86_64
      artifacts: true
  variables:
    DOCKERFILE: ci/docker/Dockerfile.app
    PERSIST_IMAGE_NAME: $CSCS_REGISTRY_PATH/appimg-x86_64:$CI_COMMIT_SHORT_SHA
    DOCKER_BUILD_ARGS: '["IMG=$BASE_IMAGE"]'

build_app_image_aarch64:
  extends: .container-builder-cscs-gh200
  stage: build_app
  needs:
    - job: build_base_image_aarch64
      artifacts: true
  variables:
    DOCKERFILE: ci/docker/Dockerfile.app
    PERSIST_IMAGE_NAME: $CSCS_REGISTRY_PATH/appimg-aarch64:$CI_COMMIT_SHORT_SHA
    DOCKER_BUILD_ARGS: '["IMG=$BASE_IMAGE"]'
```

Example: multiarch app image

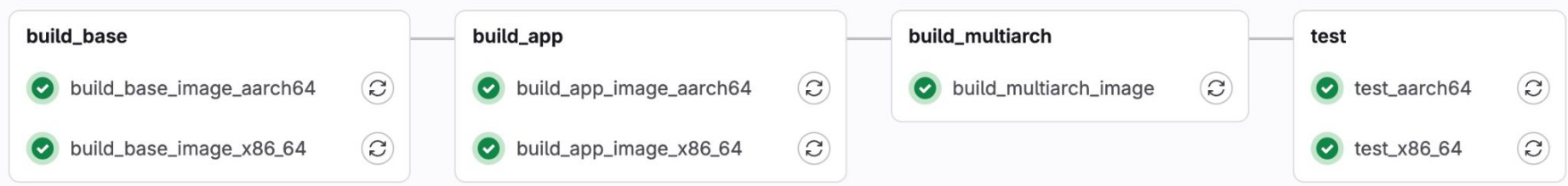
```
build_multiarch_image:
  stage: build_multiarch
  extends: .make-multiarch-image
  variables:
    PERSIST_IMAGE_NAME_X86_64:
"$CSCS_REGISTRY_PATH/appimg-x86_64:$CI_COMMIT_SHORT_SHA"
    PERSIST_IMAGE_NAME_AARCH64:
"$CSCS_REGISTRY_PATH/appimg-aarch64:$CI_COMMIT_SHORT_SHA"
    PERSIST_IMAGE_NAME: "$CSCS_REGISTRY_PATH/appimg:$CI_COMMIT_SHORT_SHA"
```

Example: run tests

```
test_x86_64:
  stage: test
  extends: .container-runner-eiger-zen2
  image: $CSCS_REGISTRY_PATH/appimg:$CI_COMMIT_SHORT_SHA
  script:
    - /helloworld/build/hello
  variables:
    SLURM_JOB_NUM_NODES: 2

test_aarch64:
  stage: test
  extends: .container-runner-daint-gh200
  image: $CSCS_REGISTRY_PATH/appimg:$CI_COMMIT_SHORT_SHA
  script:
    - /helloworld/build/hello
  variables:
    SLURM_JOB_NUM_NODES: 2
```

Example: Done



Questions?



Container builder API service

- Documentation at <https://confluence.cscs.ch/x/UQXJMw>
- Uses CSCS API gateway at <https://developer.cscs.ch>

```
# Generate access token - valid for about 5 minutes
$ ACCESS_TOKEN="$(curl -u <your-consumer-key>:<your-consumer-secret> --silent -X POST
https://auth.cscs.ch/auth/realms/firecrest-clients/protocol/openid-connect/token -d
"grant_type=client_credentials" | jq -r '.access_token')"
```

```
# helper variables
$ API="https://api.cscs.ch/ciext/v1/container/build"
$ AUTH="Authorization: Bearer $ACCESS_TOKEN"
```

```
# Build container image
$ curl -H "$AUTH" --data-binary @path/to/Dockerfile "${API}?arch=x86_64"
```

```
# Build container image in custom registry
$ curl -H "$AUTH" \
  -H "X-Registry-Username <your-dockerhub-username>" \
  -H "X-Registry-Password: <your-dockerhub-token>" \
  --data-binary @path/to/Dockerfile \
  "${API}?arch=x86_64&image=docker.io/<your-dockerhub-username>/my_image_name:latest"
```

Container builder API service

- Documentation at <https://confluence.cscs.ch/x/UQXJMw>
- Uses CSCS API gateway at <https://developer.cscs.ch>

```
# Generate access token - valid for about 5 minutes
$ ACCESS_TOKEN="$(curl -u <your-consumer-key>:<your-consumer-secret> --silent -X POST
https://auth.cscs.ch/auth/realms/firecrest-clients/protocol/openid-connect/token -d
"grant_type=client_credentials" | jq -r '.access_token')"
```

```
# helper variables
$ API="https://api.cscs.ch/ciext/v1/container/build"
$ AUTH="Authorization: Bearer $ACCESS_TOKEN"
```

```
# build container image with build context
$ cd path/to/build_context
$ tar -czf - . | curl -H "$AUTH" \
  --data-binary @- \
  "${API}?arch=x86_64&dockerfile=relative/path/to/Dockerfile"
```

- Think of this to be the command: `docker build -f relative/path/to/Dockerfile .`
- Sending the build context to the server is needed, when you have *COPY/ADD* statements in your Dockerfile

Container builder API service vs container-builder

API build

```
$ cd path/to/build_context
$ tar -czf - . | curl -H "$AUTH" \
  -H "X-Registry-Username <your-dockerhub-username>" \
  -H "X-Registry-Password: <your-dockerhub-token>" \
  --data-binary @- \
"$${API}?arch=x86_64&dockerfile=relative/path/to/Dockerfile&image=docker.io/<your-dockerhub-username>/my_image_name:latest"
```

CI yaml

```
some job name:
  extends: .container-builder-cscs-zen2
  variables:
    DOCKERFILE: relative/path/to/Dockerfile
    PERSIST_IMAGE_NAME: docker.io/<your-dockerhub-username>/my_image_name:latest
    CUSTOM_REGISTRY_USERNAME: '<your-dockerhub-username>'
    CUSTOM_REGISTRY_PASSWORD: '<your-dockerhub-token>'
```

Build context is the repository source code

UENV builder API service

- Build your own UENV
- Only works from CSCS network

```
$ uenv build /path/to/UENV/recipe my-uenv/25.04@clariden%gh200
Log          : https://cicd-ext-mw.cscs.ch/ci//uenv/build?image=cvltat9oag0s73813j00-3690753405420143
Status       : submitted

Destination
Registry     : jfrog.svc.cscs.ch/uenv
Namespace    : service
Label        : my-prgenv/25.04:1746504708@clariden%gh200

$ uenv image pull service::my-prgenv/25.04:1746504708@clariden%gh200
```