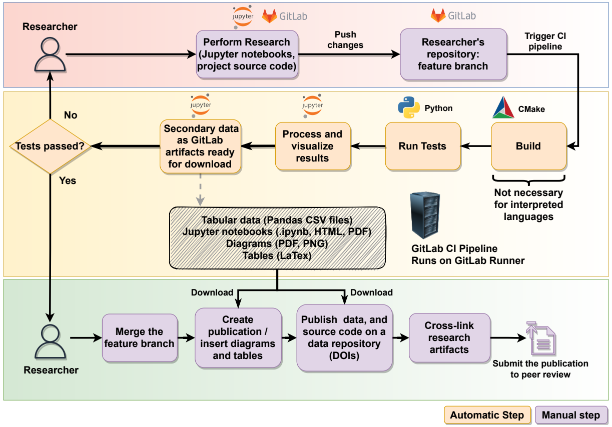
"Continuous" Integration of Scientific Software (in Computational Science and Engineering)

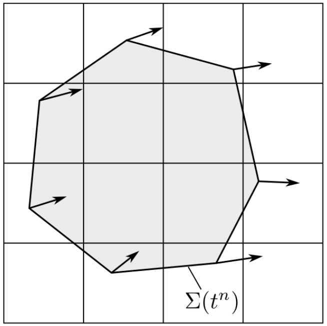
High Performance Computing in Hessen (HiPerCH)-13 2021-09-23

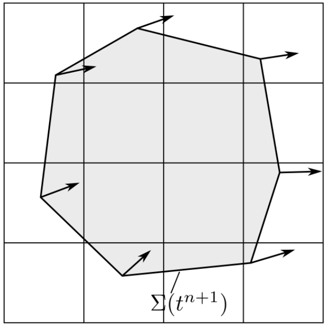


Researcher
jupyter
GitLab
Perform Research (Jupyter notebooks, project source code)
Push changes
GitLab
Researcher's repository: feature branch
Trigger CI pipeline
No
Tests passed?
Yes
jupyter
Secondary data as GitLab artifacts ready for download
jupyter
Process and visualize results
Python
Run Tests
CMake
Build
Not necessary for interpreted languages
Tabular data (Pandas CSV files)
Jupyter notebooks (.ipynb, HTML, PDF)
Diagrams (PDF, PNG)
Tables (LaTex)
GitLab CI Pipeline
Runs on GitLab Runner
Download
Download
Researcher
Merge the feature branch
Create publication / insert diagrams and tables
Publish data, and source code on a data repository (DOIs)
Cross-link research artifacts
Submit the publication to peer review
Automatic Step
Manual step

Motivation: multiphase flow simulation methods

Lagrangian / Eulerian Interface Advection (LEIA) methods





- Fluids that do not mix are separated by an interface $\Sigma(t)$ (surface in 3D).
- Goal: track $\Sigma(t)$ as it moves in time t and changes its topology.

Motivation: multiphase flow simulation software

Lagrangian / Eulerian Interface Advection (LEIA) Methods

TECHNISCHE UNIVERSITÄT DARMSTADT

LEIA methods [1, 2, 3, 4, 5] require **thorough testing**:

- **Verification** cases: evolution of $\Sigma(t)$ and two-phase flows with exact solutions.
- **Validation** with respect to experiments.
- Testing **serial and parallel computational efficiency**.

[1] Marić, T., Marschall, H., & Bothe, D. (2015). lentFoam–A hybrid Level Set/Front Tracking method on unstructured meshes. Computers & Fluids, 113, 20-31.

[2] Tolle, T., Bothe, D., & Marić, T. (2020). SAAMPLE: A Segregated Accuracy-driven Algorithm for Multiphase Pressure-Linked Equations. Computers & Fluids, 200, 104450.

[3] Marić, T., Kothe, D. B., & Bothe, D. (2020). Unstructured un-split geometrical Volume-of-Fluid methods–A review. Journal of Computational Physics, 420, 109695.

[4] Marić, T. (2021). Iterative Volume-of-Fluid interface positioning in general polyhedrons with Consecutive Cubic Spline interpolation. Journal of Computational Physics: X, 11, 100093.

[5] Tolle, T., Gründing, D., Bothe, D., & Marić, T. (2021). Computing volume fractions and signed distances from triangulated surfaces immersed in unstructured meshes. arXiv preprint arXiv:2101.08511.

Computational Science and Engineering software in university research groups

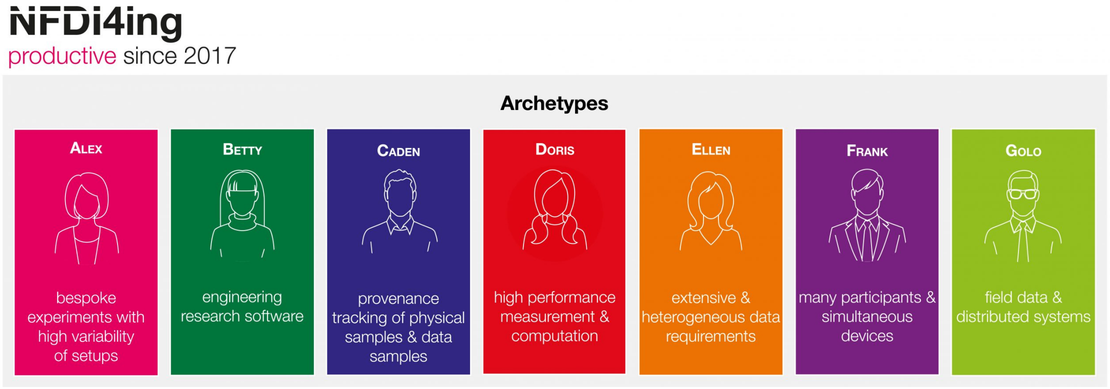
Boundary and initial conditions

- Publish or perish 🎓[6] prioritizes publications over scientific software.
- Dedicated resources for increasing software quality are usually not available.
- Ph.D. students rotate every 3-5 years, postdocs every 1-2 years.
  - Little or no overlap between successors and predecessors.
- Large-scale software design is not a mandatory part of the CSE curriculum.
  - Different CSE background: (Applied) Mathematics, Mechanical Engineering, Physics, Informatics.
- Real-world example: onboarding people into ▸ OpenFOAM module development.

[6]Symbol of a publish-or-perish simplification of the workflow :)

NFDI4Ing to the rescue!

Resources for engineering research software



NFDi4ing
productive since 2017

Archetypes

Alex	Betty	Caden	Doris	Ellen	Frank	Golo
bespoke experiments with high variability of setups	engineering research software	provenance tracking of physical samples & data samples	high performance measurement & computation	extensive & heterogeneous data requirements	many participants & simultaneous devices	field data & distributed systems

NFDI4Ing resources.

BSSW to the rescue!

Resources for engineering research software

TECHNISCHE UNIVERSITÄT DARMSTADT

https://bssw.io

better scientific software

Better Scientific Software resources.

Computational Science and Engineering software in university research groups

The chaos scientific legacy code

Betty is a CSE researcher, working with a legacy research code.
Why is Betty so (rightfully) angry?

- Betty inherited a **research software that is only partially tested**.
- Betty inherited a **research software that isn't automatically tested**.
  - Betty changes one part of the code and gets her model running, only to see 10 other things fail, after days of manually running tests.
- Betty's software has **no documentation of the scientific workflow**.
  - Betty doesn't know how to use existing scripts to run simulations and analyze (reproduce) results.
- Betty's software has **disjoint (diverging) versions** - that she can't integrate.
- **Betty can't even find code versions used to generate results in the publications from her research group.**

Computational Science and Engineering software in university research groups

The chaos of developing entirely new research software

"Après moi, le déluge" - "After me, the flood"

Louis XV of France

Research software generally does not matter, as long as papers are published (🎓).

Missed opportunities

- *Finding results* made easy by cross-linking code versions, data and publications.
- *Faster extension / combination of existing ideas* if their respective versions are integrated.
- *Faster comparison of results* with previous ideas automating verification / validation.
- *Automatic reproducibility* of results using automated testing and version control.
- *Faster onboarding* with documented scientific verification and validation workflows.

Computational Science and Engineering software in university research groups

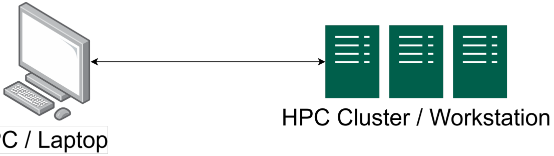
Continuous integration and cross-linking to the rescue

Automated testing (verification and validation), **version control**, and **cross-linking** reports, source code and research data increase Findability, Accessibility and Reproducibility (FAIR) and **speed up research.**

- **Continuous Integration (CI) = automatic testing + version control.**
- CSE research requires **scientific workflows**: initialize simulations, run parameter variations, agglomerate data, visualize, and check results.
- CI can be used to **automate and document scientific workflows**.
- CI ensures that the **integration of new changes does not break existing functionality**.
- Once the changes are integrated, the publication, the source code and the data are published on pre-print and data repositories and **cross-linked** using git tags and DOIs.

Continuous Integration of Scientific Software

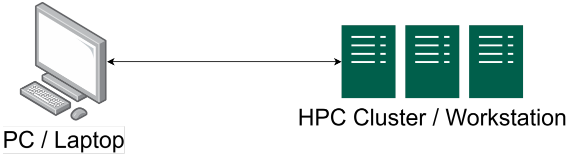
Research Software Workflow I



```
while Results are unsatisfactory do
    Work on algorithms.
    (Compile the code.)
    for All studies do
        Prepare the study.
        Run the study.
        Analyze results.
        Move results to a report.
    end for
    Compare old and new results.
end while
```

Continuous Integration of Scientific Software

Research Software Workflow II

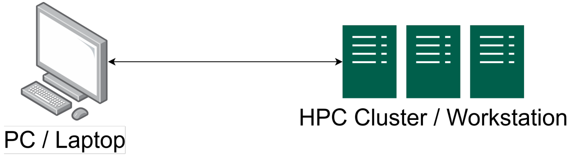


Issues...

- Starting studies takes time.
- Analyzing results takes time.
- Often the results are not checked "live" as the study runs - **waste of research time and CPUh**.
- **Only the researcher knows the details** behind the initialization, running and post-processing scripts - **when this person leaves, the reproducibility is gone.**
- A researcher may forget to run a study and believe all tests have passed.

Continuous Integration of Scientific Software

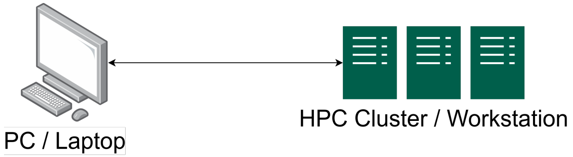
Automating the research workflow I



```
while Results are unsatisfactory do
    Work on algorithms.
    (Compile the code.)
    for All studies do
        Prepare the study.
        Run the study.
        Analyze results.
        Move results to a report.
    end for
    Compare old and new results.
end while
```

Continuous Integration of Scientific Software

Automating the research workflow I



```
while Results are unsatisfactory do
    Work on algorithms.
    (Compile the code.)
    Run initialization scripts (jobs).
    Run simulation scripts (jobs).
    (Run postprocessing scripts (jobs)).
    Visualize results live in Jupyter notebooks.
end while
```

Continuous Integration of Scientific Software

Automating the research workflow II

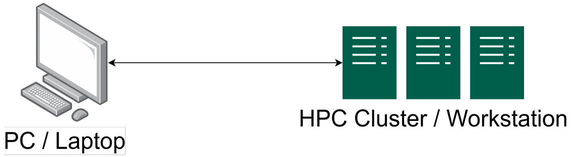
Manual steps of the research workflow,

```
(Compile the code.)
for All studies do
    Prepare the study.
    Run the study.
    Analyze results.
    Move results to a report.
end for
Compare old and new results.
```

are now automated using scripts **that do not require additional knowledge / input (metadata).**

Continuous Integration of Scientific Software

Automating the research workflow III

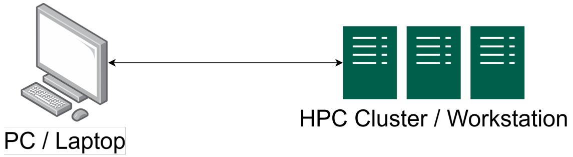


1. The **new** results are satisfactory.
2. Similar automated workflows are executed for existing tests.
3. All results are checked.
4. The milestone has been reached, the version can be integrated.

Works well manually when there aren't many previous verification/validation tests and their analysis is relatively simple.
Are we sure that we ran all the tests and examined the results properly?

Continuous Integration of Scientific Software

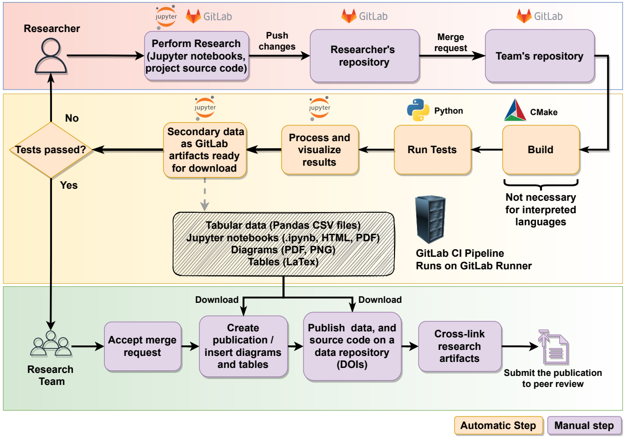
Automating the research workflow IV



- **Manual testing takes a lot of time**.
- **Manual testing** of all previous tests **is prone to error** - even if V&V scripts do not require metadata.
- Relevant **V&V tests are automated using Continuous Integration (CI)**.
  - Changes are pushed to the upstream version control repository.
  - The remote repository starts the so-called **CI** test pipeline (a sequence of tests).
  - Tests are automatically run, processed and visualized.

(Continuous) Integration of scientific software

Schematic diagram for the team workflow

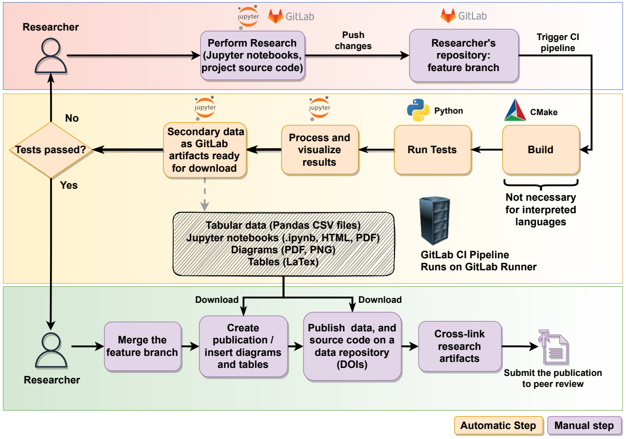


Researcher
GitLab
Perform Research (Jupyter notebooks, project source code)
Push changes
Researcher's repository
Merge request
Team's repository
Python
CMake
No
Tests passed?
Secondary data as GitLab artifacts ready for download
Process and visualize results
Run Tests
Build
Yes
Not necessary for interpreted languages
Tabular data (Pandas CSV files)
Jupyter notebooks (.ipynb, HTML, PDF)
Diagrams (PDF, PNG)
Tables (LaTex)
GitLab CI Pipeline
Runs on GitLab Runner
Download
Download
Accept merge request
Create publication / insert diagrams and tables
Publish data, and source code on a data repository (DOIs)
Cross-link research artifacts
Submit the publication to peer review
Research Team
Automatic Step
Manual step

Working in a team.

(Continuous) Integration of scientific software

Schematic diagram for the individual workflow



Researcher
GitLab
Perform Research (Jupyter notebooks, project source code)
Push changes
GitLab
Researcher's repository: feature branch
Trigger CI pipeline
No
Tests passed?
Yes
Secondary data as GitLab artifacts ready for download
Process and visualize results
Python
Run Tests
CMake
Build
Not necessary for interpreted languages
Tabular data (Pandas CSV files)
Jupyter notebooks (.ipynb, HTML, PDF)
Diagrams (PDF, PNG)
Tables (LaTex)
GitLab CI Pipeline
Runs on GitLab Runner
Download
Download
Merge the feature branch
Create publication / insert diagrams and tables
Publish data, and source code on a data repository (DOIs)
Cross-link research artifacts
Submit the publication to peer review
Researcher
Automatic Step
Manual step

Working alone.

(Continuous) Integration of scientific software
CI in a nutshell I

- A **text (YAML) file** is added to a repository, that **specifies the tests** (jobs) in a CI pipeline.
- When the YAML file is pushed to an upstream git repository (GitLab), **GitLab creates a CI pipeline from the YAML file**.
- The CI pipeline needs a **machine for running tests - the GitLab runner**.
  - Shared runners on gitlab.com have limited capacity.
  - We can install and register our own GitLab runner.
- **A Docker image encapsulates the computing environment**.
  - **Virtualization/Containerisation** increases reproducibility and simplifies testing.
- The Docker image must be publicly accessible for it to be used by a shared runner.

(Continuous) Integration of scientific software

CI in a nutshell II

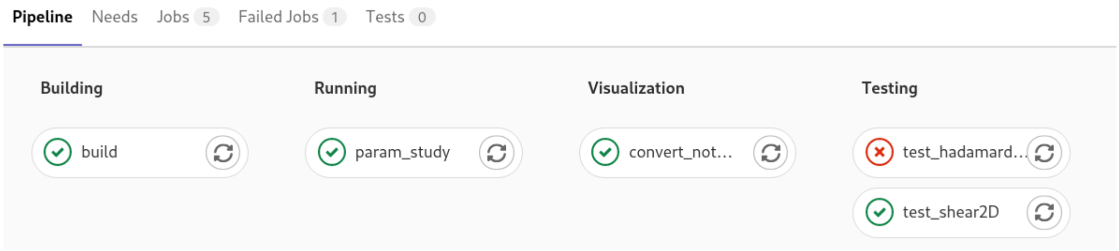
```
initialization_param_study:
  stage: running
  dependencies:
    - build_release
  script:
    # run the parameter variation tests
    - cd cases/initialization/3dinit
    - ./create_and_run_levelset.sh
    - ./reproduce_publication_results.sh
  artifacts:
    paths:
        - cases/initialization/3dinit/*.csv
        - cases/initialization/3dinit/*.pdf
```

Example YAML file

- The **CI pipeline starts the right scripts in the right order**: it documents the research workflow.
- A click of a button in a web browser reproduces results for any version of the research software.
- Continuous **integration** is used to **integrate** only those changes that improve the software and don't break existing tests.

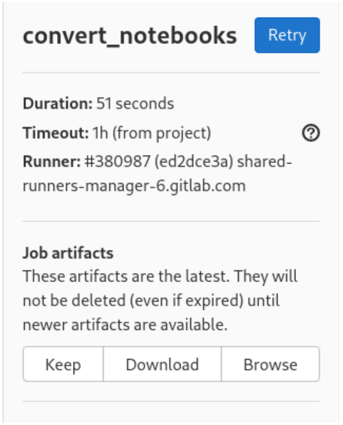
(Continuous) Integration of scientific software
CI in a nutshell III

An example CI pipeline



(Continuous) Integration of scientific software

CI in a nutshell IV


convert_notebooks

Duration: 51 seconds

Timeout: 1h (from project)

Runner: #380987 (ed2dce3a) shared-runners-manager-6.gitlab.com

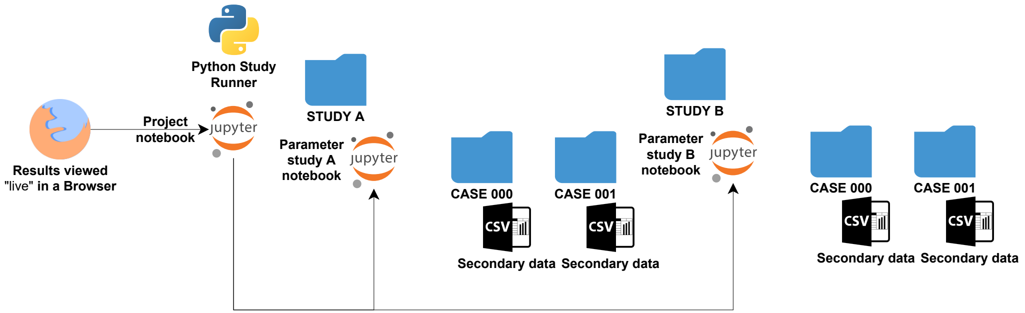
Job artifacts

These artifacts are the latest. They will not be deleted (even if expired) until newer artifacts are available.

- Files created within a CI job are gone when the job ends.
- GitLab uses **job artifacts** to pass on data from one job to the next.
- **Job artifacts can only be files stored in project's sub-folders.**
- Libraries and applications are passed to other jobs as artifacts.
- **Artifacts can be downloaded on the GitLab project website.**

(Continuous) Integration of scientific software

Running tests I



Organize your simulation studies.

(Continuous) Integration of scientific software
Running tests II

TECHNISCHE UNIVERSITÄT DARMSTADT

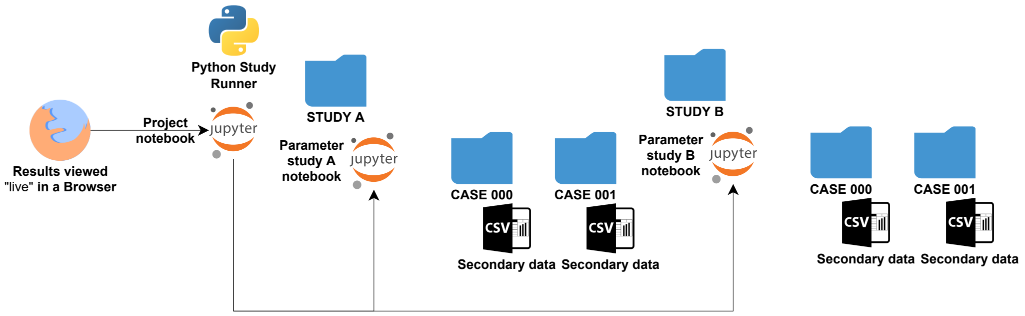
- Success of CSE methods is measured using verification and validation data.
- Effective comparison with others (previous versions) hinges on data organization.
- **Goal:** easily **programmatically identify** parameters used in a simulation case.
- **Legacy code**:
  - use the existing folder structure and parameterization tools
  - The mapping (case000) $\rightarrow$ (parameter vector) must be stored (YAML, ...)
- **New code**:
  1. Simple folder and file structure
  2. HDF5[7] or other open data format.
  3. Alternative to HDF5: **ExDir**[8]

[7] https://www.hdfgroup.org/solutions/hdf5

[8] Dragly, Svenn-Arne, et al. "Experimental Directory Structure (Exdir): An alternative to HDF5 without introducing a new file format." Frontiers in neuroinformatics 12 (2018): 16.

(Continuous) Integration of scientific software

Running tests II



- Associate simulation cases with their metadata.
- `{case000 : {N_CELLS: 32, MODEL : shear2D}}`
- Store this information using a standard open-source format (**Interoperability** in FAIR).

(Continuous) Integration of scientific software

Running tests III

Use Jupyter notebooks[9] and pandas[10] for

- **Documentation**: geometry, initial and boundary conditions, error norms, comparison data.
- **Data processing**: verification errors (conservation, convergence, stability), validation errors
- **Result analysis**: interactive and remote, while simulations are running!
    - Jupyter notebook server can be started on a remote machine and accessed in a web browser.

[9] https://jupyter.org/
[10] https://pandas.pydata.org/

(Continuous) Integration of scientific software
Processing and visualizing tests

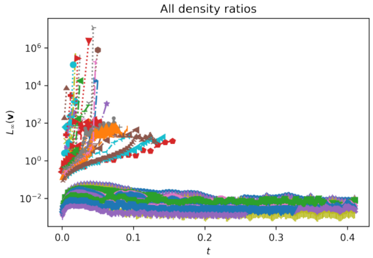
```
jupyter nbconvert notebook.ipynb --execute --to FORMAT
```

- Agglomerate secondary data into `pandas.MultiIndex` CSV files.
- Run each jupyter notebook in the repository.
- Export secondary data and notebooks in different formats as artifacts.
- **Visualization**
  - Download the artifact and open the notebook 🎓.
  - Notebooks contain information on failing tests.
  - Mapping "caseID" $\rightarrow$ "parameters" is crucial for re-starting failed parameter variations!

(Continuous) Integration of scientific software

Secondary data I

- Data used for diagrams and tables in a publication.
- Data we compare our results with.
- Data we waste time scanning from (sometimes our own) publications in CSE.



(a) Old inconsistent method: interface stable only for cases with density ratio $\rho^- / \rho^+ = 1$

Imagine scanning this diagram.

Preprint: https://arxiv.org/abs/2109.01595

Data: https://doi.org/10.48328/tudatalib-627

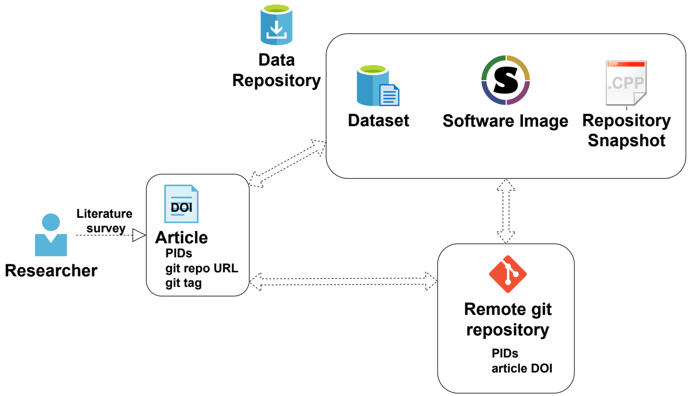
(Continuous) Integration of scientific software
Secondary data II

pandas.MultiIndex CSV with metadata for secondary data

- `pandas.MultiIndex` saved in "metadata columns".
- **Metadata is repeated**: not an issue for the small secondary data!
- Metadata in columns $\rightarrow$ `pandas.MultiIndex` $\rightarrow$ strongly simplified data analysis.
- **Direct readable export of tables to LaTeX!**

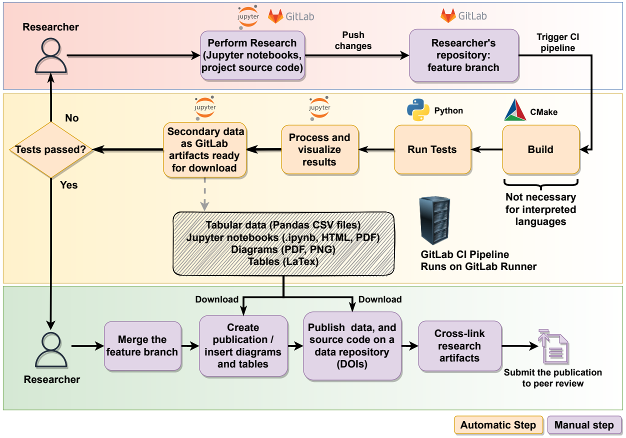
	H	L_INF	O(L_INF)	EPSILON_R_EXACT_MAX	O(EPSILON_R_EXACT_MAX)
VELOCITY_MODEL					
SHEAR_2D	0.125000	0.032961	1.833407	0.032961	1.833407
SHEAR_2D	0.062500	0.009249	1.955529	0.009249	1.955529
SHEAR_2D	0.031250	0.002385	1.988745	0.002385	1.988745
SHEAR_2D	0.015625	0.000601	1.997178	0.000601	1.997178
SHEAR_2D	0.007813	0.000150	1.999294	0.000150	1.999294
SHEAR_2D	0.003906	0.000038	1.999294	0.000038	1.999294

(Continuous) Integration of scientific software
Cross-linking I



(Continuous) Integration of scientific software

Schematic diagram for the individual workflow



Researcher
jupyter
GitLab
Perform Research (Jupyter notebooks, project source code)
Push changes
GitLab
Researcher's repository: feature branch
Trigger CI pipeline
No
Tests passed?
Yes
jupyter
Secondary data as GitLab artifacts ready for download
jupyter
Process and visualize results
Python
Run Tests
CMake
Build
Not necessary for interpreted languages
Tabular data (Pandas CSV files)
Jupyter notebooks (.ipynb, HTML, PDF)
Diagrams (PDF, PNG)
Tables (LaTex)
GitLab CI Pipeline
Runs on GitLab Runner
Download
Download
Researcher
Merge the feature branch
Create publication / insert diagrams and tables
Publish data, and source code on a data repository (DOIs)
Cross-link research artifacts
Submit the publication to peer review
Automatic Step
Manual step

Working alone.

(Continuous) Integration of scientific software
Cross-linking II

Cross-linking is done manually.

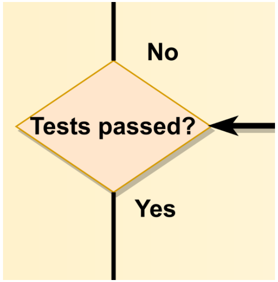
- Place whatever you can under version control.
- **When a set of milestones is reached (*release*)** , use git-tags as version snapshots, and upload the research data to a data repository, e.g. TUDatalib at TU Darmstadt, or Zenodo.
  - Secondary data (diagrams, tables), raw data (simulations, experiments), archive of the research software, ...
- Data uploaded to a data repository is associated with Persistent Identifiers (PIDs), e.g. DOIs.
- Cite the research data using DOIs in the report (article, preprint).
- Upload the report to a pre-print repository, e.g. ArXiv.
- Edit the data on the data repository and mention the arXivID.
- Submit the pre-print to a journal for peer-review.

Continuous Integration of Scientific Software
Cross-linking III

- **Research software is compared with existing publications.**
- A major milestone are improved results for a set of verification / validation tests.
- The cross-linking therefore revolves around the publication (pre-print, report, ...).
- The cross-linking makes it possible to find the version of research software used to generate the results in the publication: repository link + git tag, repository snapshot, software image.
- Once the version is found, CI automatically reproduces all results from the publication with a click of a button.

(Continuous Integration with result visualization)

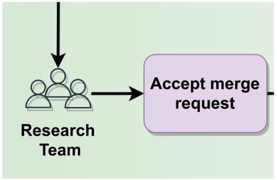
Test evaluation I



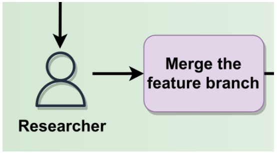
Straightforward for easily quantifiable errors

- **Examples:** volume conservation, order of convergence, total wall clock time, weak scaling, ...
- **Python scripts test secondary data** agglomerated by Jupyter notebooks from simulation results.

(Continuous Integration with result visualization)
Test evaluation II



or

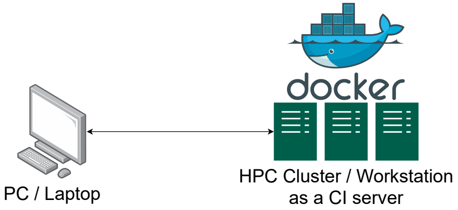


Difficult for errors that cannot be quantified easily

- **Examples:**
  - Is is the difference between simulation and experiment data $\leq 4\%$?
  - How to quantify the difference for complex signals?
- **Option 1: Researchers evaluate the test results even if all CI jobs pass.**
  - A simple and efficient solution 🎓.
- **Option 2: Use statistics to quantify the difference.**

(Continuous) Integration of scientific software

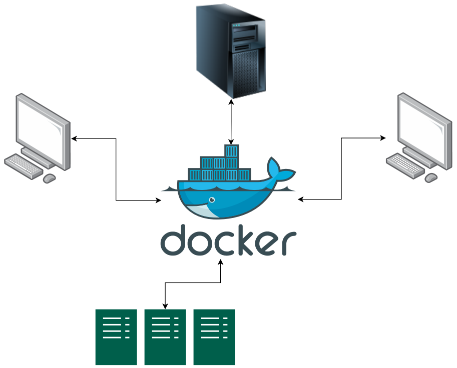
Docker (containerization)



- Instead of installing the research software only on the laptop/PC and the HPC cluster / workstation, we install it in a virtual environment - **a Docker image**.
- The Docker image then works on any machine that runs Docker.
- Sharing research software becomes trivial - if our colleague wants to use our software, no installation (besides Docker) is required.

(Continuous) Integration of scientific software

Docker (containerization)



- Instead of installing the research software only on the laptop/PC and the HPC cluster / workstation, we install it in a virtual environment - **a Docker image**.
- The Docker image then works on any machine that runs Docker.
- Sharing research software becomes trivial - if our colleague wants to use our software, no installation (besides Docker) is required.

(Continuous) Integration of scientific software

Computing resources

The GitLab CI requires a **GitLab runner: a machine that runs the CI jobs.**

1. **Short few CPU-core tests**: work-PC 🎓.
2. **Short many-core tests**: obtain a workstation with a 64-Core CPU[11]🎓.
3. **HPC tests**: combine 1. or 2. with an HPC cluster.

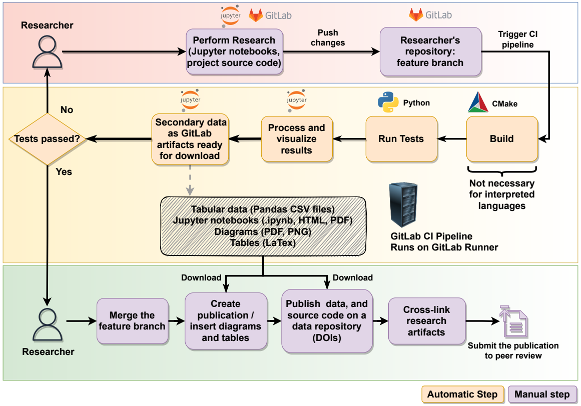
An HPC cluster is relevant for production tests and performance measurements.

- This workflow uses coarse ("smoke") tests 🎓
  - Unit tests run for 1. and 2.
  - Convergence ensured for 1. and 2.
  - Is efficient in parallel for 1. and 2.
- **Challenge**: Is it possible to combine 1., 2. and 3. and publish instead of perish 🎓?

[11]Thanks to CRC 1194 at TU Darmstadt.

(Continuous) Integration of scientific software

Summary



Researcher
GitLab
Perform Research (Jupyter notebooks, project source code)
Push changes
Researcher's repository: feature branch
Trigger CI pipeline
No
Tests passed?
Yes
Secondary data as GitLab artifacts ready for download
Process and visualize results
Python
Run Tests
CMake
Build
Not necessary for interpreted languages
Tabular data (Pandas CSV files)
Jupyter notebooks (.ipynb, HTML, PDF)
Diagrams (PDF, PNG)
Tables (LaTex)
GitLab CI Pipeline
Runs on GitLab Runner
Download
Download
Researcher
Merge the feature branch
Create publication / insert diagrams and tables
Publish data, and source code on a data repository (DOIs)
Cross-link research artifacts
Submit the publication to peer review
Automatic Step
Manual step

```
1:  Track changes using version-control.
2:  while Milestone not reached do
3:      for study in studies do                    ▷ On an HPC cluster.
4:          Automate data processing and visualization.
5:          Run study.
6:          Check results and apply code changes.
7:      end for
8:      if results are improved on the HPC cluster then
9:          Push changes to the remote repository.
10:         if CI pipeline tests pass then
11:             Milestone reached.
12:             Add new tests to the CI pipeline,
13:             Merge feature into development branch.
14:             Cross-link publication, data, and source code.
15:         end if
16:     end if
17: end while
```

(Continuous) Integration of scientific software

Similarity with other workflows / best practices

Our *(subjective)* estimates* of similarity $1 - 5$ (higher is more similar), $-$: aspect not addressed.

DOI	*Branching model*	*TDD*	*Cross-linking*	*CI*	(Meta)data standardization
10.12688/f1000research.11407.1	-	-	-	-	1
10.3934/math.2016.3.261	-	-	-	-	2
10.1371/journal.pbio.1001745	1	2	-	-	-
10.1371/journal.pcbi.1005510	-	-	3	1	3
10.1145/2723872.2723881	1	-	-	1	-
10.1145/3324989.3325719	1	-	-	5	-
10.1371/journal.pone.0230557	1	-	-	1	4
10.1145/3219104.3219147	1	-	-	4	-

**The list may still be incomplete.*

(Continuous) Integration of scientific software
Hands on: overview

1. Repository preparation

A minimal repository representing an exemplary "status quo".

2. Create a Docker image

Configure a reproducible testing environment.

3. Define CI pipeline through a YAML file

Define tests, how and when they are executed and what results to store.

4. Setup your own GitLab runner

Provide a machine for execution of tests.

(Continuous) Integration of scientific software

Hands on: prepare the example repository

Create your own copy of the example repository by forking:

- Log in to https://gitlab.com/.
- Go to https://gitlab.com/tmaric/minimal-cse-ci-examples.
- Click **fork** (upper right corner).
- Select a namespace, e.g. your personal one.
- Select either *Private* or *Public* as visibility level, both are fine.
- Click **Fork project**.
- Clone your fork on your machine:

  ```
  ?> git clone your-fork-URL
  ```

(Continuous) Integration of scientific software

Hands on: overview

1. Repository preparation

A minimal repository representing an exemplary "status quo".

2. Create a Docker image

Configure a reproducible testing environment.

3. Define CI pipeline through a YAML file

Define tests, how and when they are executed and what results to store.

4. Setup your own GitLab runner

Provide a machine for execution of tests.

(Continuous) Integration of scientific software

Hands on: install Docker I

Specific steps depend on your Linux distribution (Docker documentation)
Here for Ubuntu Focal:

1. `?> sudo apt-get update`
2. `?> sudo apt-get install apt-transport-https ca-certificates curl gnupg lsb-release`
3.
```
?> curl -fsSL https://download.docker.com/linux/ubuntu/gpg \
 | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg
```
4.
```
?> echo \
"deb [arch=amd64 signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] \
https://download.docker.com/linux/ubuntu \
$(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```
5. `?> sudo apt-get update`
6. `?> sudo apt-get install docker-ce docker-ce-cli containerd.io`

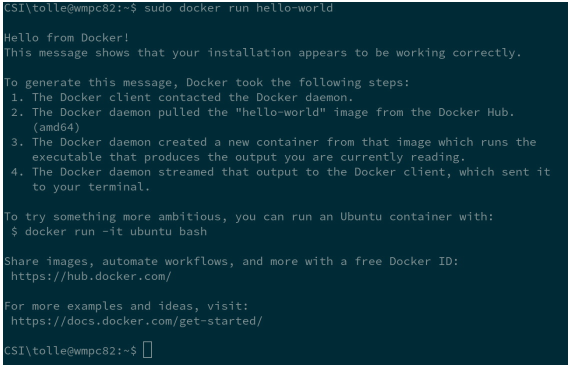
(Continuous) Integration of scientific software

Hands on: install Docker II

Check your Docker installation by running

- `?> sudo docker run hello-world`

The output should look as shown on the right.



```
CSI\tolle@wmpc82:~$ sudo docker run hello-world

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/

CSI\tolle@wmpc82:~$ 
```

(Continuous) Integration of scientific software

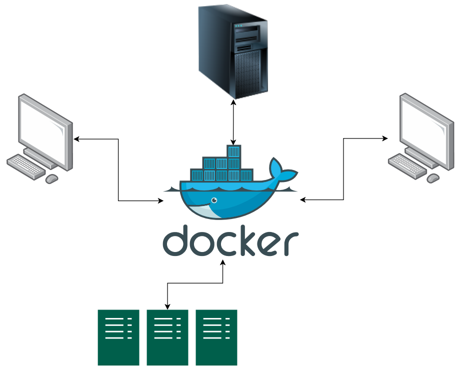
Hands on: creating a Docker image

In the **`minimal-cse-ci-examples`** repository

```
?> git checkout starting-point
?> git checkout -b feature/dockerfile
```

(Continuous) Integration of scientific software

Hands on: creating a Docker image I



- Docker images are computing environments that contain (**dependencies**) needed to build the research software, run simulations and process results.
- Sharing docker images removes the need to install the dependencies on different machines.
- The computing environment in a Docker image is usually based on an existing Linux distribution.
- The Docker image is built from a text file, that specifies installation steps for the dependencies, the so-called **Dockerfile**.

(Continuous) Integration of scientific software

Hands on: creating a Docker image II

Base Docker image

```
FROM ubuntu:focal

# Set timezone
RUN apt-get update --fix-missing && \
    DEBIAN_FRONTEND="noninteractive" apt-get -y install tzdata
```

- We'll use Ubuntu 20.04 (focal) as the base system.
- Steps that are usually done manually (setting the timezone) are automated.

(Continuous) Integration of scientific software

Hands on: creating a Docker image III

TECHNISCHE UNIVERSITÄT DARMSTADT

Dependency installation

```
# Install packages
RUN apt update && apt-get install --fix-missing -y \
    # Building
    build-essential cmake \
    # Version control
    git \
    # Python
    python3 \
    # Visualization
    python3-matplotlib python3-numpy \
    # Data analysis
    python3-pandas \
    # Test visualization
    jupyter-notebook jupyter-nbconvert \
    # Debugging the image
    vim
```

- **RUN** runs commands in the **Docker container**.
- The **Docker container** is a process spawned using the Docker image as the computing environment.
- Install the software needed for the scientific workflow (dependencies).

(Continuous) Integration of scientific software

Hands on: creating a Docker image IV

Software setup

```
## Default Ubuntu to python3
RUN update-alternatives --install \
    /usr/bin/python python /usr/bin/python3 10
```

Some specifics

- Alternative (**g++**) compiler.
- Alternative working directory.

(Continuous) Integration of scientific software

Hands on: creating a Docker image V

TECHNISCHE UNIVERSITÄT DARMSTADT

Complete Dockerfile for the minimal example

```
FROM ubuntu:focal

# Set timezone
RUN apt-get update --fix-missing && \
    DEBIAN_FRONTEND="noninteractive" apt-get -y install tzdata

# Install packages
RUN apt update && apt-get install --fix-missing -y \
    # Building
    build-essential cmake \
    # Version control
    git \
    # Python
    python3 \
    # Visualization
    python3-matplotlib python3-numpy \
    # Data analysis
    python3-pandas \
    # Test visualization
    jupyter-notebook jupyter-nbconvert \
    # Debugging the image
    vim

## Default Ubuntu to python3
RUN update-alternatives --install \
    /usr/bin/python python /usr/bin/python3 10
```

- The example Dockerfile installs all dependencies for the minimal example on Ubuntu 20.04.
- The installation commands would be different for another operating system.
- A more complex software (e.g. OpenFOAM) requires a larger Dockerfile.
- This lets us define the computing environments that are supported by the research software.

(Continuous) Integration of scientific software

Hands on: creating a Docker image VI

Building the image

```
?> sudo docker build . \
    -f minimal-cse-ci-dockerfile\_ubuntu-focal \
    -t minimal-cse-ci-dockerfile\_ubuntu-focal
```

- "`.`": current directory
- "-f" name of the Dockerfile (defaults to "Dockerfile")
- "-t" tag (name) of the Docker image

(Continuous) Integration of scientific software

Hands on: creating a Docker image VII

Listing Docker images

```
?> sudo docker image list
REPOSITORY                             TAG    IMAGE    ID   CREATED        SIZE
minimal-cse-ci-dockerfile_ubuntu-focal latest 921233ec4b44 9 minutes ago 982MB
```

- The image is built on the machine (host) where the **`docker build`** command is called.
- Docker uses a so-called **image registry** to store images.
- For Continuous Integration the images are built on the machine where the tests are run or shared on **Dockerhub**.

(Continuous) Integration of scientific software

Hands on: creating a Docker image VIII

"**Spinning a container**" (running a Docker image)

```
?> sudo docker run -it minimal-cse-ci-dockerfile_ubuntu-focal /bin/bash
root@b2c14ee0fd58:/# ls
bin  boot  dev  etc  home  lib  lib32  lib64  libx32  media  mnt
opt  proc  root  run  sbin  srv  sys  tmp  usr  var
root@b2c14ee0fd58:/# cd
root@b2c14ee0fd58:~# pwd
/root
```

- The container behaves just like a "regular" Ubuntu.
- **Jobs (test) commands for the Continuous Integration are checked/debugged inside a running container.**
  - Forgot to install a dependency.
  - The research software does not compile with installed dependencies.
  - ...

(Continuous) Integration of scientific software

Hands on: creating a Docker image IX

Working within the container : compiling the software

```
?> git clone https://gitlab.com/tmaric/minimal-cse-ci-examples.git
?> cd minimal-cse-ci-examples && mkdir build && cd build
?> cmake .. && make
?> ./myapp
```

The same steps will be done in the Docker container by the Continuous Integration

- Clone the repo.
- Build the software.
- Run the tests.

(Continuous) Integration of scientific software

Hands on: creating a Docker image X

Analyzing the data using Jupyter notebooks

```
?> cd ..
?> jupyter nbconvert --execute mynotebook.ipynb --to html
```

- On the cluster, one would start the Jupyter notebook server and connect to it locally.
- Here the notebook is used to process the results and visualize secondary data as tables and diagrams.

(Continuous) Integration of scientific software

Hands on: creating a Docker image XI

Extracting the data from the container:

- Find the ID of the container you're on (execute on your machine)

  ```
  sudo docker ps
  ```

- Copy the results from the container onto the local machine (execute on your machine)

  ```
  mkdir container-data
  sudo docker cp f2dff55edf7a:/root/minimal-cse-ci-examples \
      container-data/
  ```

- Examine the data and the Jupyter notebook in a browser.

Note: the sequence f2dff55edf7a is system dependent ID, so it'll be different for you.

(Continuous) Integration of scientific software

Hands on: creating a Docker image XII

- Saving the container or an image as a tar file

```
sudo docker commit f2dff55edf7a test:latest
```

- You can exit/close the container by pressing Ctrl+d.
- View the newly create image with 'name:tag' using

```
sudo docker image list

REPOSITORY TAG     IMAGE ID      CREATED               SIZE
test       latest  f2dff55edf7a  About a minute ago    983MB
```

- Save the image into a tar file

```
sudo docker save test:latest -o container-archive.tar
```

- Load an image into Docker's registry to work with it

```
docker load < container-archive.tar
```

(Continuous) Integration of scientific software

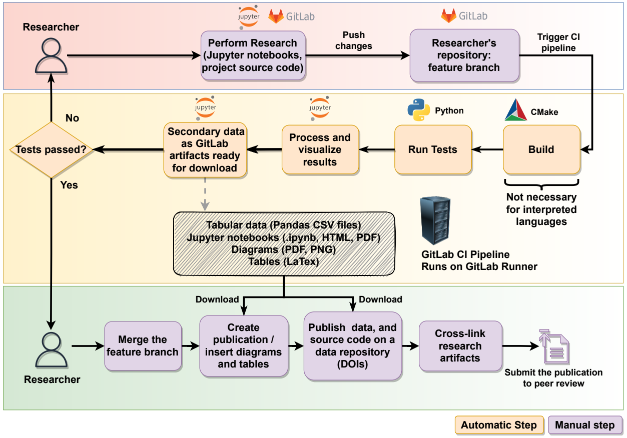
Hands on: creating a Docker image XIII

- Usually, the Docker image "lives" locally on the test machine.
- However, it can also be shared publicly on Dockerhub, for example (don't do this now)

```
?> docker login
?> docker tag name:tag username/name:tag
?> docker push username/name:tag
```

- This image can now be used by everyone.
- Note: once you exit/stop a container all data/files created inside the container are discarded.

(Continuous) Integration of scientific software
Hands on: creating a Docker image XIV


Researcher
jupyter
GitLab
Perform Research (Jupyter notebooks, project source code)
Push changes
GitLab
Researcher's repository: feature branch
Trigger CI pipeline
No
jupyter
jupyter
Python
CMake
Tests passed?
Secondary data as GitLab artifacts ready for download
Process and visualize results
Run Tests
Build
Yes
Not necessary for interpreted languages
Tabular data (Pandas CSV files)
Jupyter notebooks (.ipynb, HTML, PDF)
Diagrams (PDF, PNG)
Tables (LaTex)
GitLab CI Pipeline
Runs on GitLab Runner
Download
Download
Merge the feature branch
Create publication / insert diagrams and tables
Publish data, and source code on a data repository (DOIs)
Cross-link research artifacts
Submit the publication to peer review
Researcher
Automatic Step
Manual step

All the steps done so far manually using Docker, namely,

1. **building** the scientific software,
2. **running tests**,
3. **processing data**
4. **exporting** the data and Jupyter notebooks,

are automated by Continuous Integration, that uses Docker for encapsulating the computing environment.

(Continuous) Integration of scientific software

Hands on: overview

TECHNISCHE UNIVERSITÄT DARMSTADT

1. Repository preparation

A minimal repository representing an exemplary "status quo".

2. Create a Docker image

Configure a reproducible testing environment.

3. Define CI pipeline through a YAML file

Define tests, how and when they are executed and what results to store.

4. Setup your own GitLab runner

Provide a machine for execution of tests.

(Continuous) Integration of scientific software

Hands on: enabling CI for a GitLab project

In the **`minimal-cse-ci-examples`** repository

```
    ?> git checkout added-dockerfile
    ?> git checkout -b feature/enable-ci
```

(Continuous) Integration of scientific software

Hands on: enabling CI for a GitLab project I

- Adding the **`.gitlab-ci.yml`** file your project and pushing the change to the GitLab remote repo configures the CI pipeline.
- The YAML file specifies the Docker image that is used for testing

```yaml
image: "tmaric/minimal-cse-ci:ubuntu-focal"

stages:
    - building
    - running
    - visualization
```

- and the so-called job **stages**: collections of jobs for building, running tests and visualization.
- For example, the **building** stage may multiple jobs, building the software for
  - production,
  - debugging,
  - performance measurements.

(Continuous) Integration of scientific software

Hands on: enabling CI for a GitLab project II

- The building stage in the YAML file defines build jobs like this one

```
build:
  stage: building
  script:
      - git clone https://gitlab.com/tmaric/minimal-cse-ci-examples.git
      - cd minimal-cse-ci-examples && mkdir build && cd build
      - cmake ..
      - make

  artifacts:
    paths:
        - minimal-cse-ci-examples/mynotebook.ipynb
        - minimal-cse-ci-examples/build/myapp
```

- where the repository is cloned and built with specific options.
- For example `cmake -DCMAKE_BUILD_TYPE=Debug` can set up the build for debugging.
- **artifacts** are downloadable files passed on to other jobs.

(Continuous) Integration of scientific software

Hands on: enabling CI for a GitLab project III

- The running stage in the YAML file defines how simulations (studies) run

```
param_study:
  stage: running
  dependencies:
    - build

  script:
    - cd minimal-cse-ci-examples/build && ./myapp

  artifacts:
    paths:
        - minimal-cse-ci-examples/mynotebook.ipynb
        - minimal-cse-ci-examples/build/myapp
        - minimal-cse-ci-examples/build/poly-data.csv
```

- Without a successful **build**, simulations do not run.
- Here the artifacts are the secondary data and the notebooks that visualize them.

(Continuous) Integration of scientific software

Hands on: enabling CI for a GitLab project IV

- The **visualization** stage in the YAML file saves time by converting Jupyter notebooks

```
convert_notebooks:
  stage: visualization
  dependencies:
    - param_study

  script:
    - cd minimal-cse-ci-examples
    - jupyter nbconvert mynotebook.ipynb --execute --to html

  artifacts:
    paths:
        - minimal-cse-ci-examples/mynotebook.*
        - minimal-cse-ci-examples/build/myapp
        - minimal-cse-ci-examples/build/polydata.csv
```

- HTML is easiest, other formats are available (PDF, markdown,...).
- HTML notebooks can be viewed in the browser.

(Continuous) Integration of scientific software

Hands on: enabling CI for a GitLab project V

Final **.gitlab-ci.yml** file.

Lessons learned I

- Defining artifacts path starts at **`your-project/`**.
- YAML files require debugging:
  - syntax: use GitLab's CI Lint tool
  - everything else: the only way to do this effectively is to commit changes and push them upstream.
- It is possible to partially debug locally using

  ```
  gitlab-runner exec docker job-name
  ```

  but this does not work with artifacts and dependencies.

(Continuous) Integration of scientific software
Hands on: enabling CI for a GitLab project VI

Final **.gitlab-ci.yml** file.

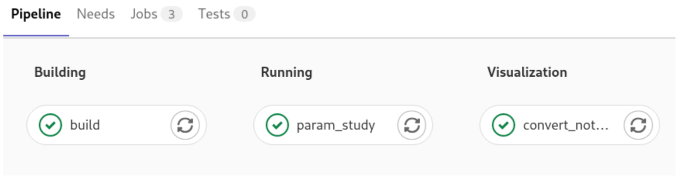
Lessons learned II

- Generally, and for the CI, scripts that reproduce data without requiring input for the users speed up work.
  `simulation-directory > ./reproduce-density-ratio-data`
- It takes time to set up the CI, but it pays off in debugging time as problems are found automatically.
- Exporting **`*.ipynb`** jupyter notebooks and their data

(Continuous) Integration of scientific software

Hands on: enabling CI for a GitLab project VII

The CI pipeline of the Minimal Working Example (MWE) repository



(Continuous) Integration of scientific software

Hands on: overview

1. Repository preparation

A minimal repository representing an exemplary "status quo".

2. Create a Docker image

Configure a reproducible testing environment.

3. Define CI pipeline through a YAML file

Define tests, how and when they are executed and what results to store.

4. Setup your own GitLab runner

Provide a machine for execution of tests.

(Continuous) Integration of scientific software

Hands on: setup a GitLab runner I (why a self-managed runner)

An incomplete comparison:

Self-managed runner

- No shared runners available
- Provided CI/CD minutes of plan insufficient (e.g. 400 min per month for GitLab's free plan)
- Control over hardware and runner configuration

Shared runners

- No need to provide "always on" hardware
- No need for maintenance

Overall: require less of your time

(Continuous) Integration of scientific software

Hands on: setup a GitLab runner II (install GitLab runner)

Using instructions from GitLab documentation for Ubuntu:

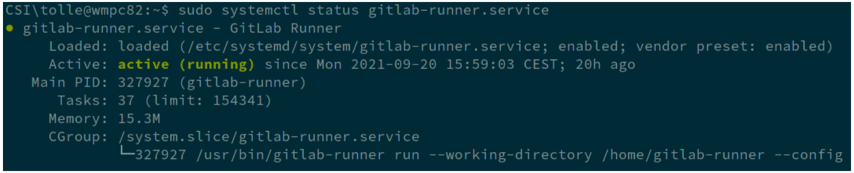
1.
   ```
   ?> curl -L \
   "https://packages.gitlab.com/install/repositories/runner/gitlab-runner/script.deb.sh" \
    | sudo bash
   ```

2.
   ```
   ?> sudo apt-get install gitlab-runner
   ```

Check the status of the runner:

- `?> sudo systemctl status gitlab-runner.service`

The output should indicate that it is active:



```
CSI\tolle@wmpc82:~$ sudo systemctl status gitlab-runner.service
● gitlab-runner.service - GitLab Runner
     Loaded: loaded (/etc/systemd/system/gitlab-runner.service; enabled; vendor preset: enabled)
     Active: active (running) since Mon 2021-09-20 15:59:03 CEST; 20h ago
   Main PID: 327927 (gitlab-runner)
      Tasks: 37 (limit: 154341)
     Memory: 15.3M
     CGroup: /system.slice/gitlab-runner.service
             └─327927 /usr/bin/gitlab-runner run --working-directory /home/gitlab-runner --config
```

(Continuous) Integration of scientific software

Hands on: setup a GitLab runner III (register GitLab runner)

Follow GitLab documentation on how to register a runner:

- Obtain a token for project-specific runner: go to your fork of the *minimal-cse-ci-examples* on gitlab.com and then to **Settings > CI/CD** and expand the **Runners** sections.
- There you find a section **Specific runners** and aforementioned token.

Register your runner (instructions for Linux):

- `?> sudo gitlab-runner register`

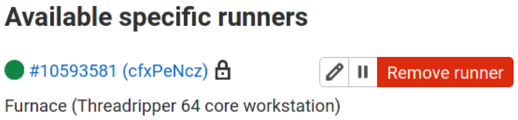
You need to provide some information regarding your runner, e.g. your project's token. See next slide.

(Continuous) Integration of scientific software

Hands on: setup a GitLab runner IV (register GitLab runner)

Option	Value
GitLab instance URL	https://gitlab.com/
Token	Obtained from the project on GitLab, see previous slide
Runner description	describe the machine used as runner, useful to distinguish multiple runners
Tags	leave empty, not required here. Useful for advanced pipelines
Runner executor	docker (see here for comparison of executors.)
Default image	Because we chose docker as executor: name of the default Docker image

You should now see your runner under
Available specific runners:


Available specific runners

● #10593581 (cfxPeNcz)

Furnace (Threadripper 64 core workstation)

Acknowledgements

TECHNISCHE UNIVERSITÄT DARMSTADT

CRC 1194

Interaction between Transport and Wetting Processes

Funded by the German Research Foundation (DFG) – Project-ID 265191195 – CRC 1194 : Z-INF