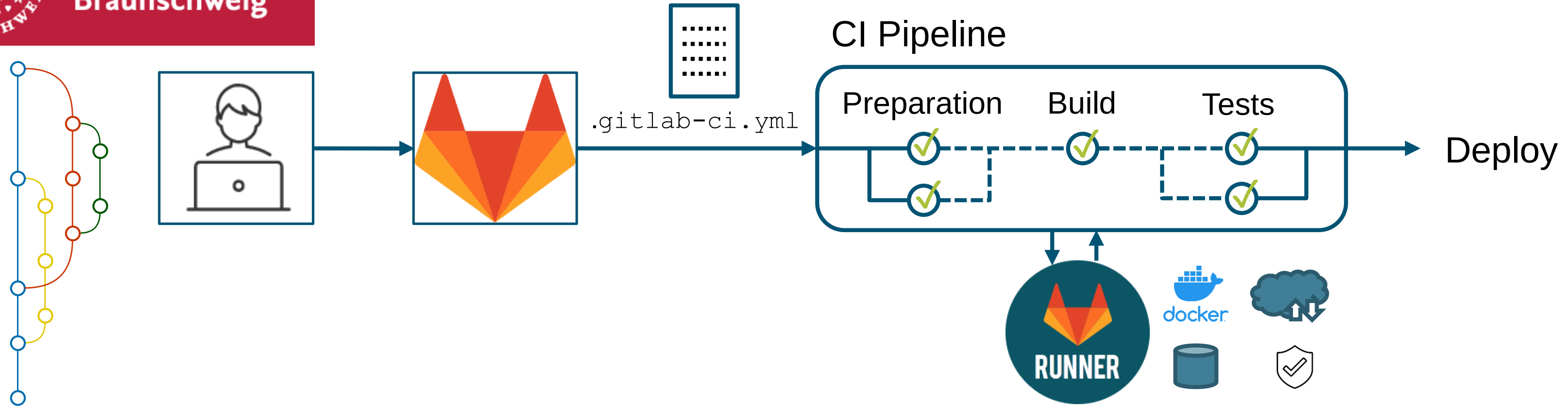


Suresoft workshops series

Introduction to Continuous Integration (CI) using GitLab and Containerization

Harikrishnan Sreekumar and Lucas Hermann, 5th September 2022

Workshop objectives

- Get familiarized with Containerization and Continuous Integration
- Create a Docker image using Docker and host the image in GitLab
- Establish a CI pipeline in GitLab for a software project

Workshop agenda

Introduction to Suresoft

Part 1: Containerization using Docker

- Introduction to container technologies
- Hands-on exercise: Build a custom Docker image + Break
- Hands-on exercise: Host an image in GitLab

Part 2: Continuous Integration (CI) using GitLab

- Introduction to CI
- Hands-on exercise: Create a simple CI pipeline + Break
- Hands-on exercise: Manage pipeline artifacts and code coverage + Break

Part 3: Demonstration of elPaSo CI pipeline and containerization approaches

Information

- Workshop slides and documentation (more details, commands, hints, …)

  https://suresoft.gitlab-pages.rz.tu-bs.de/workshop-website
- We look forward to your questions and experiences – please unmute and interrupt anytime during the workshop or post in chat
- Workshop preparation – see in workshop documentation
  - Visual Studio Code
  - Git installation
  - Example code project (please fork the project once again)
  - Docker
- We use the main room for our hands-on session – no break-out rooms
- We use python as our standard language

Introduction to Suresoft

Who are we?

18 People from 7 Institutes and Facilities

INSTITUT FÜR AKUSTIK

Institute of Operating Systems and Computer Networks

IfN Institut für Nachrichtentechnik

iRMB

Institut für Physikalische und Theoretische Chemie

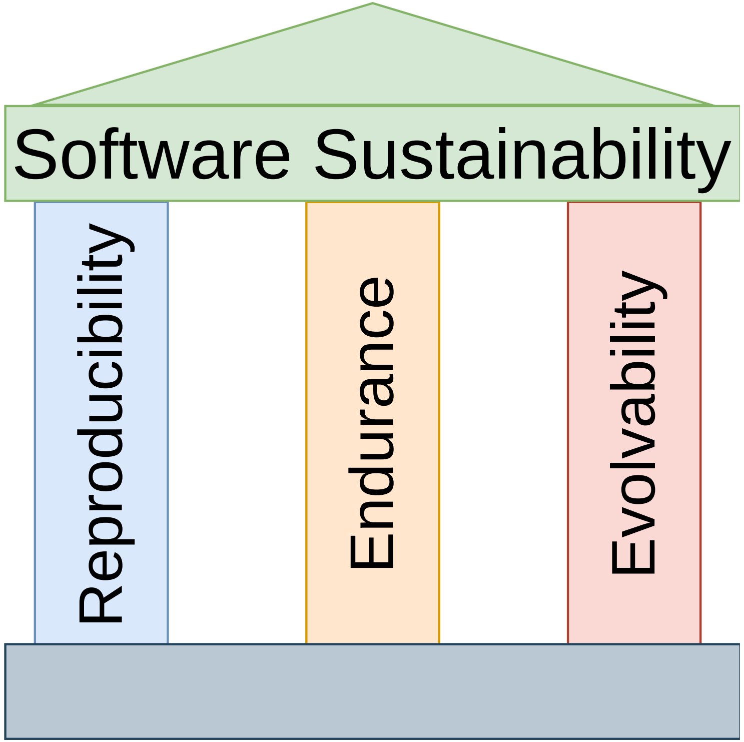
Technische Universität Braunschweig

University Library & Gauß-IT-Zentrum

Common problems of research software

1. Software has low code quality
2. Software is neither published nor documented
3. Software depends on a specific runtime environment (e.g third party libraries), which may not be available to other researchers

Software sustainability



SURESOFT Approach for Sustainable Software

Education

- Documentation
- Software Engineering Principles
- Testing

Infrastructure & Methods

- Version Control
- Archiving & Publication
- CI & Automated Testing
- Virtualization
- Issue Reporting
- Installation & Deployment

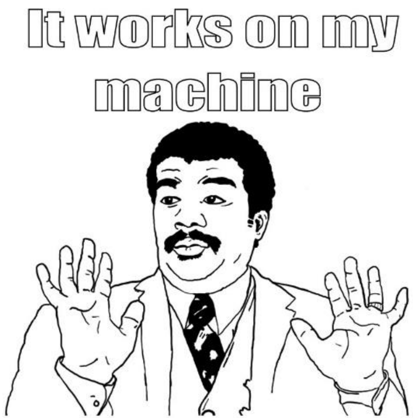
Suresoft workshop series

Every 4 weeks

1.	Version Control using Git	June 13
2.	Clean Code and Refactoring	July 11
3.	Introduction to Software Testing	August 8
4.	**Introduction to Continuous Integration (CI) using GitLab and Containerization**	**September 5**
5.	Principles of Software Engineering	October 10
6.	Introduction to Design Patterns	November 7
7.	Working with legacy code	TBA
8.	Test Driven Development	TBA
9.	Documentation	TBA

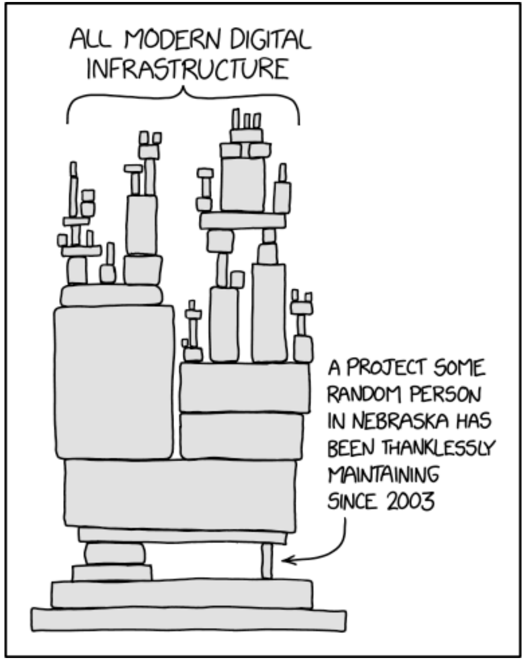
Part 1: Containerization

Motivation



[blogs.sap.com]

Motivation | Dependency Hell

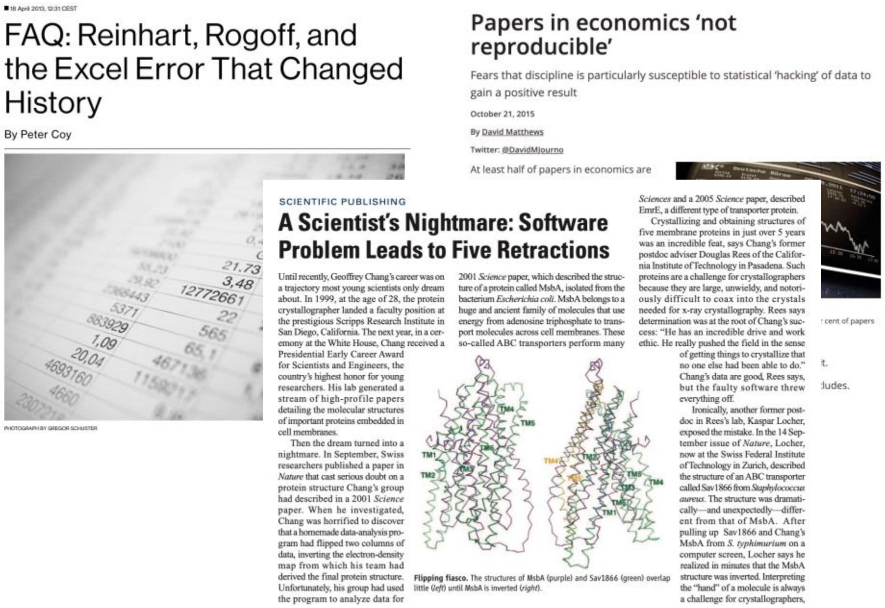


[xkcd.com]

[Matt Rickard:The nine circles of dependency hell (and a roadmap out). 2021]

Motivation | Credibility crisis

Questionable reliability, accuracy, reproducibility and verifiability of the results …



18 April 2013, 12:31 CEST

FAQ: Reinhart, Rogoff, and the Excel Error That Changed History

By Peter Coy

PHOTOGRAPH BY GREGOR SCHUSTER

Papers in economics 'not reproducible'

Fears that discipline is particularly susceptible to statistical 'hacking' of data to gain a positive result

October 21, 2015

By David Matthews

Twitter: @DavidMJourno

At least half of papers in economics are

SCIENTIFIC PUBLISHING

A Scientist's Nightmare: Software Problem Leads to Five Retractions

Until recently, Geoffrey Chang's career was on a trajectory most young scientists only dream about. In 1999, at the age of 28, the protein crystallographer landed a faculty position at the prestigious Scripps Research Institute in San Diego, California. The next year, in a ceremony at the White House, Chang received a Presidential Early Career Award for Scientists and Engineers, the country's highest honor for young researchers. His lab generated a stream of high-profile papers detailing the molecular structures of important proteins embedded in cell membranes.

Then the dream turned into a nightmare. In September, Swiss researchers published a paper in *Nature* that cast serious doubt on a protein structure Chang's group had described in a 2001 *Science* paper. When he investigated, Chang was horrified to discover that a homemade data-analysis program had flipped two columns of data, inverting the electron-density map from which his team had derived the final protein structure. Unfortunately, his group had used the program to analyze data for

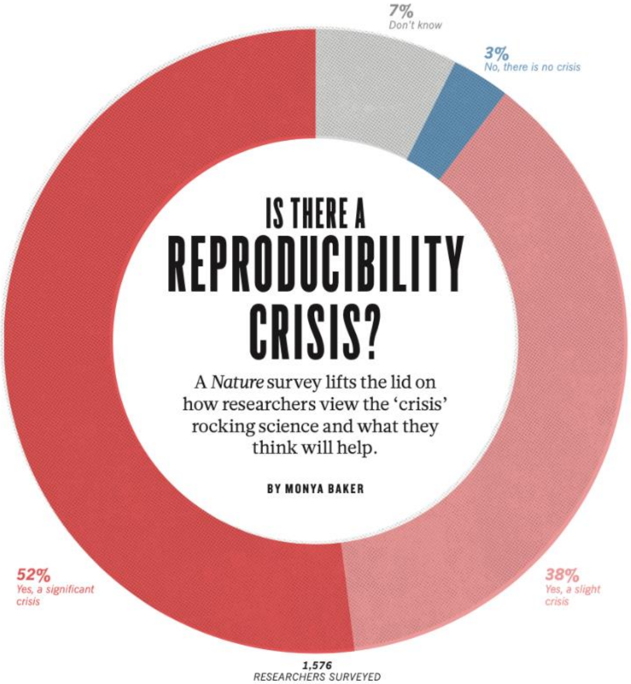
2001 *Science* paper, which described the structure of a protein called MsbA, isolated from the bacterium *Escherichia coli*. MsbA belongs to a huge and ancient family of molecules that use energy from adenosine triphosphate to transport molecules across cell membranes. These so-called ABC transporters perform many

Flipping fiasco. The structures of MsbA (purple) and Sav1866 (green) overlap little (*left*) until MsbA is inverted (*right*).

Sciences and a 2005 *Science* paper, described EmrE, a different type of transporter protein.

Crystallizing and obtaining structures of five membrane proteins in just over 5 years was an incredible feat, says Chang's former postdoc adviser Douglas Rees of the California Institute of Technology in Pasadena. Such proteins are a challenge for crystallographers because they are large, unwieldy, and notoriously difficult to coax into the crystals needed for x-ray crystallography. Rees says determination was at the root of Chang's success: "He has an incredible drive and work ethic. He really pushed the field in the sense of getting things to crystallize that no one else had been able to do." Chang's data are good, Rees says, but the faulty software threw everything off.

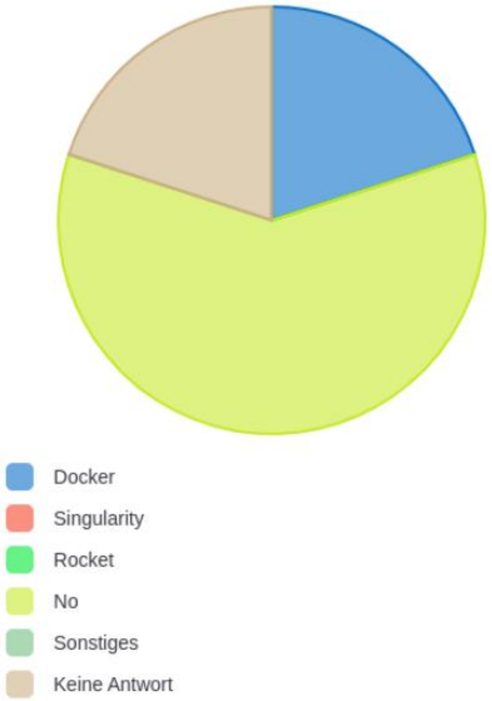
Ironically, another former postdoc in Rees's lab, Kaspar Locher, exposed the mistake. In the 14 September issue of *Nature*, Locher, now at the Swiss Federal Institute of Technology in Zurich, described the structure of an ABC transporter called Sav1866 from *Staphylococcus aureus*. The structure was dramatically—and unexpectedly—different from that of MsbA. After pulling up Sav1866 and Chang's MsbA from *S. typhimurium* on a computer screen, Locher says he realized in minutes that the MsbA structure was inverted. Interpreting the "hand" of a molecule is always a challenge for crystallographers,



https://go.nature.com/2DgtDKR

Motivation | Suresoft survey

Do you use any containerization platform?



62.07% do not rely on containers

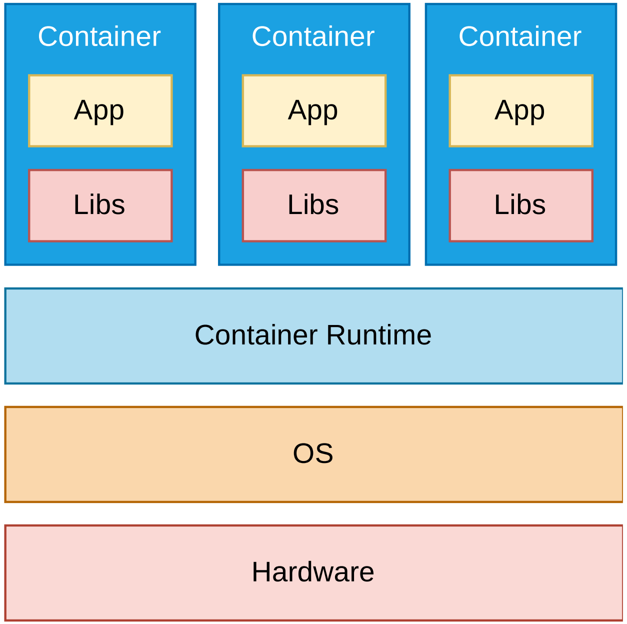
What is a container?

"A container is a standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another."

[docker.com]

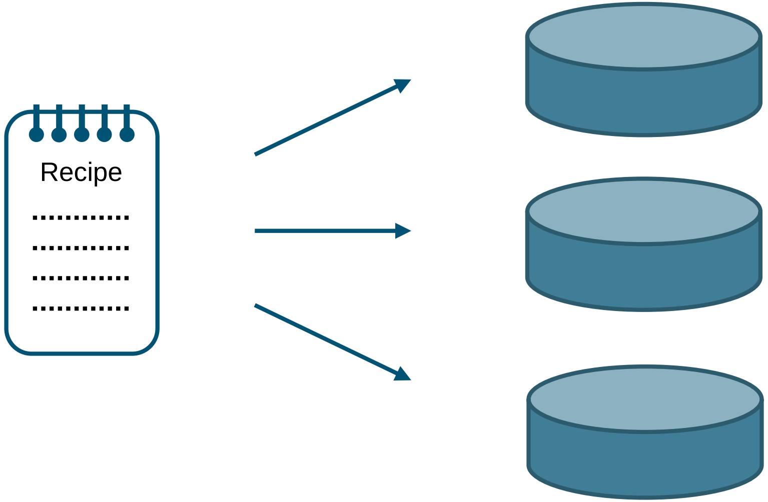
Container technologies

- Encapsulate entire runtime environment, including dependencies
- Easy to share and use Ensures reproducibility
- Scripted environment provides basic documentation
- Great for continuous integration
- Unlike running VMs, running numerous containers is possible
- Docker in CI, Singularity in HPC



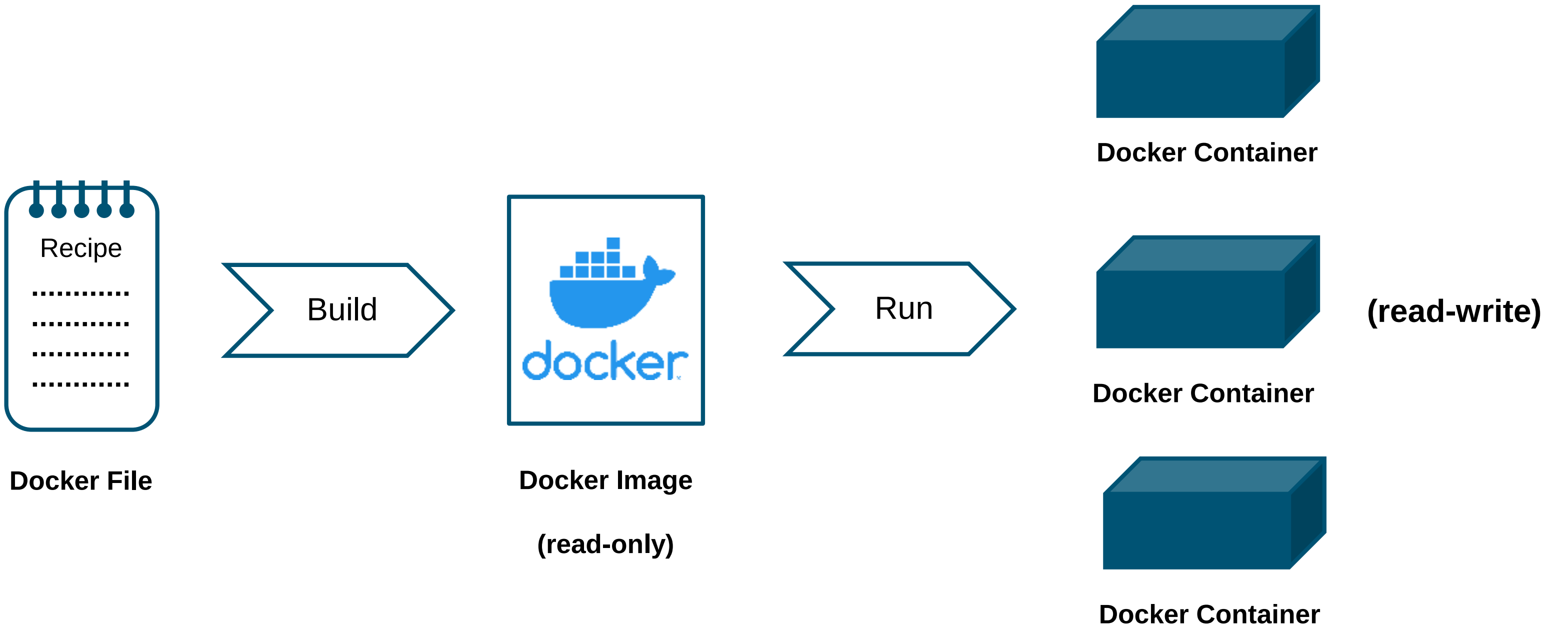
Container vs Image

- Image is a blueprint or recipe with instructions for deploying containers
- Container is a running instance of an image – a lightweight VM



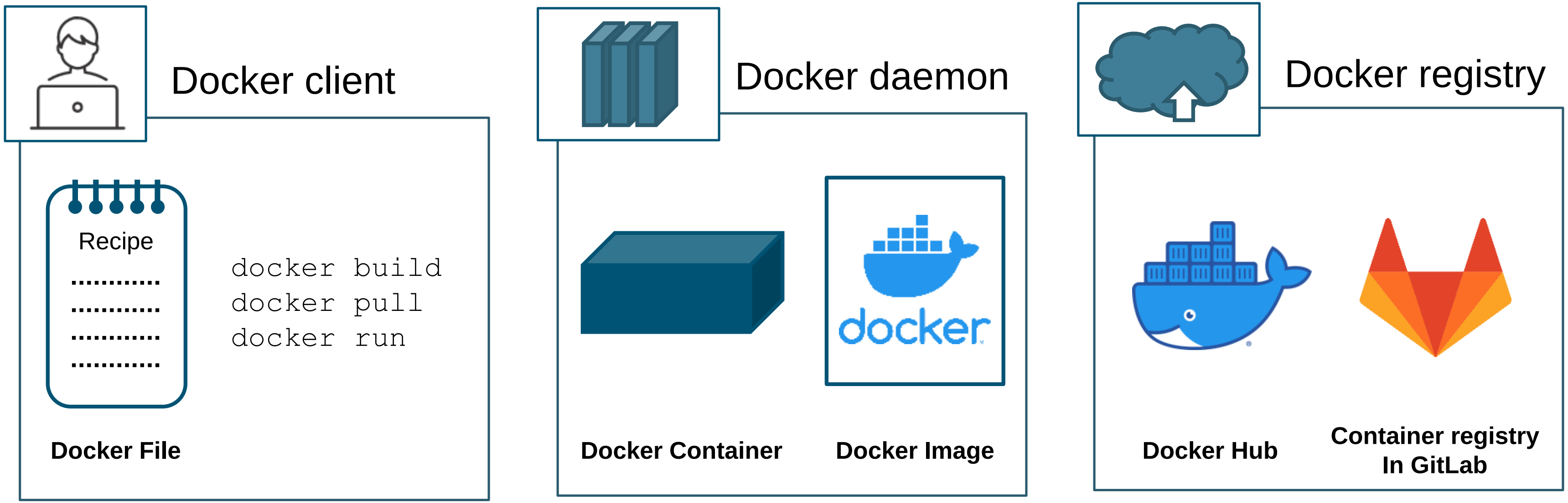
Container vs Image

- Image is a blueprint or recipe with instructions for deploying containers
- Container is a running instance of an image – a lightweight VM



How does it work?

Client-server architecture



How an user interact with Docker daemon

Manages Docker services, Docker objects (containers, images, volumes, …)

Central place to store Docker images

Docker (Containerization) for reproducible research

Dependency hell and code rot
- Handling 3rd party libraries – their compatibility, evolution, long term preservation

→ **With containerization (Docker)**
Docker image wraps the software with all the software dependencies and environment

Imprecise documentation
- Lack of documentation on how to prepare and build software dependencies and environment

→ **With containerization (Docker)**
Dockefile scripts resolves imprecise documentation

Barriers to adoption and reuse in existing solutions
- Challenges faced by researchers when adopting techniques and tools to address reproducibility

→ **With containerization (Docker)**
Easy sharing of images and integration into local development environments

[Boettiger, Carl. "An introduction to Docker for reproducible research." ACM SIGOPS Operating Systems. Review 49.1 (2015): 71-79]

Using Docker for normal applications

- Use Docker as version control for the entire software's environment
- Instead of sharing just codes, share Docker images which are ready to use

- When not to consider Docker:
  - Performance concerns → use Singularity instead of Docker for HPC
  - Security considerations
  - GUI applications → Docker prefer console based execution → PyQt framework in a Linux container is possible

[docker.com]

Using Singularity for HPC applications

- “Singularity and Docker are great friends”
- Develop with Docker and when in HPC use Singularity-ized Docker image
- Host or Hybrid MPI model – MPI implementation on the host is used by singularity to launch MPI inside the container (require compatible MPI installation)

```
mpirun -n <NUMBER_OF_RANKS> singularity exec <PATH/TO/MY/IMAGE>
                </PATH/TO/BINARY/WITHIN/CONTAINER>
```

- Singularity is currently supported in the TU BS Phoenix Cluster

[https://docs.sylabs.io/guides/3.10/user-guide/mpi.html]

[sylabs.io]

How to get Docker?

- Docker Desktop (Mac, Windows, Linux) - https://docs.docker.com/get-docker/
- Ubuntu - https://docs.docker.com/engine/install/ubuntu/

- For windows machines, you may require Hyper-V and WSL activated. Also, virtualization support enabled in the BIOS.

How to ship Docker images?

- Docker registry – Docker Hub, GitLab container registry, …
- Save and load docker images as tar using

```
docker save --output archive.tar <my-image>
docker load --input archive.tar
```

Part 1: Hands-on exercises | Create and host container

Hands-on exercise: Containerization with Docker

Technical Prerequisitities:

Install Docker on your machine (docs.docker.com/engine/install)

Make sure you have a GitLab Access Token defined and saved (https://git.rz.tu-bs.de/-/profile/personal_access_tokens)

If possible, fork the Suresoft repository and git clone it to your machine (https://git.rz.tu-bs.de/suresoft/ci-workshop-example)

Hands-on exercise: Containerization with Docker

Why use Containers?

- Docker is just one example of a containerization tool.
- A container can be compared to a very fast Virtual Machine
- The local OS and Python package versions etc. don‘t need to be touched to run code for specific versions
- Your code and also the complete environment can easily be ported to another machine on any other OS

Hands-on exercise: Containerization with Docker

Part 1: Installation

- Docker Desktop (GUI) vs. Docker Engine (CLI)
- Follow instructions on docs.docker.com/engine/install
- Test the installation in the CLI:

```
$ docker run hello-world

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

Hands-on exercise: Containerization with Docker

Part 2: Get a basic Python container

- Docker provides pre-installed images at „Docker Hub“
- Pull the latest image for python:

```
$ docker pull python
```

- Now you can use docker run and the image to run a single script inside a Python Container:

```
$ docker run -it --rm --name my-running-script -v "$PWD":/usr/src/myapp -w /usr/src/myapp python:3 python
SamplePythonScript.py
```

Hands-on exercise: Containerization with Docker

Part 3: Larger Projects and Dockerfiles

- If your project involves more than only one script, it makes sense to write a Dockerfile.
- Specific to your needs, docker builds a container image based on the Dockerfile

```
FROM python:3

WORKDIR /usr/src/exampleDir

COPY requirements.txt ./
RUN pip install --no-cache-dir -r requirements.txt

COPY . .

CMD [ "python", "./SamplePythonScript.py" ]
```

Hands-on exercise: Containerization with Docker

Part 3: Larger Projects and Dockerfiles

- Within the requirements.txt, all necessary python packages are defined
- It can be generated in your local environment using pip3 freeze

```
$ pip3 freeze > requirements.txt
```

→

```
asn1crypto==0.24.0
atpublic==0.5
attrs==17.4.0
beniget==0.4.1
cryptography==2.1.4
cycler==0.11.0
Cython==0.29.30
fenics==2019.1.0
fenics-dijitso==2019.1.0
fenics-dolfin==2019.1.0
fenics-ffc==2019.1.0.post0
fenics-fiat==2019.1.0
fenics-ufl==2019.1.0
...
...
```

Hands-on exercise: Containerization with Docker

Part 4: Building the Image

- cd into the project directory, in which the Dockerfile and the requirements.txt lie
- Make sure the Dockerfile has no file extension such as .txt
- Build the new custom image and tag it with a name:

```
$ docker build -t example-project .
```

- Under the name „example-project“ there is now a container image with your specifications

Hands-on exercise: Containerization with Docker

Part 5: Integration with VS Code

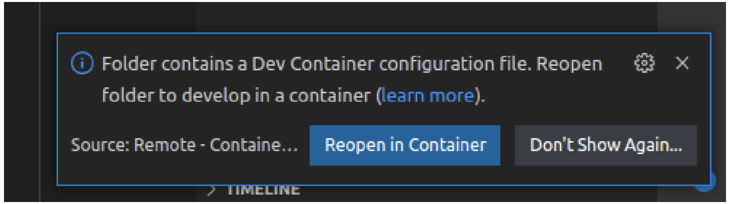
- Install the Remote-Containers extension
- Add /.devcontainer/devcontainer.json to the directory:

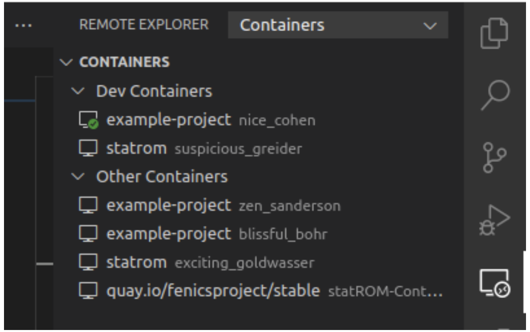
```
{
    "image": "example-project"
}
```

Hands-on exercise: Containerization with Docker

Part 5: Integration with VS Code

- The next time VS Code is opened, it asks to „reopen the folder in a container“. Click Reopen.
- In the Remote Explorer menu, it is now visible that the project runs inside a container based on your „example-project“ image.



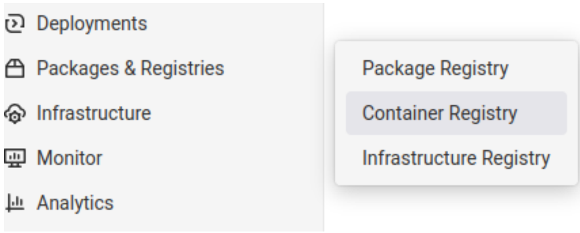


- You can now install extensions to the container and work with it as usual.

Hands-on exercise: Containerization with Docker

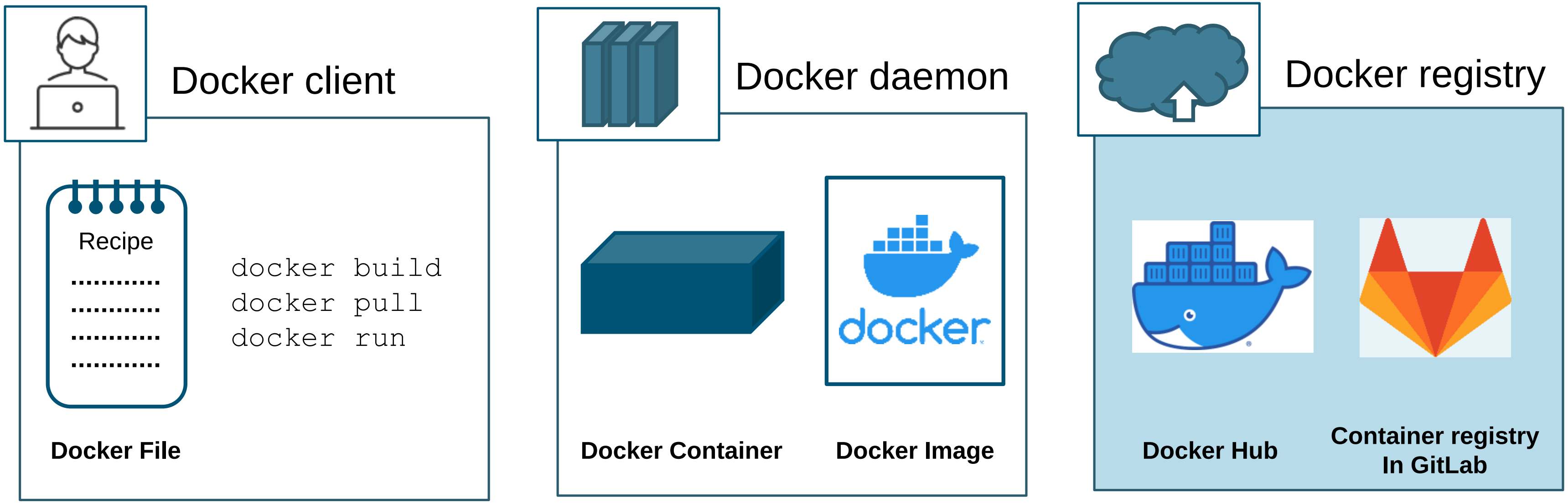
Part 6: Host the image on GitLab

- In order to perform automated tests on the image, it is useful to host in on GitLab,
- First, fork our repository: https://git.rz.tu-bs.de/suresoft/ci-workshop-example
- In the new repository, navigate to the Container Registry



Hands-on exercise: Containerization with Docker

Client-server architecture



How an user interact with Docker daemon

Manages Docker services, Docker objects (containers, images, volumes, …)

Central place to store Docker images

Hands-on exercise: Containerization with Docker

Part 6: Host the image on GitLab

- Follow the steps on the site
- Generate Access Token

There are no container images stored for this project

With the Container Registry, every project can have its own space to store its Docker images. More Information

CLI Commands

If you are not already logged in, you need to authenticate to the Container Registry by using your GitLab username and password. If you have Two-Factor Authentication enabled, use a Personal Access Token instead of a password.

```
docker login git.rz.tu-bs.de:4567
```

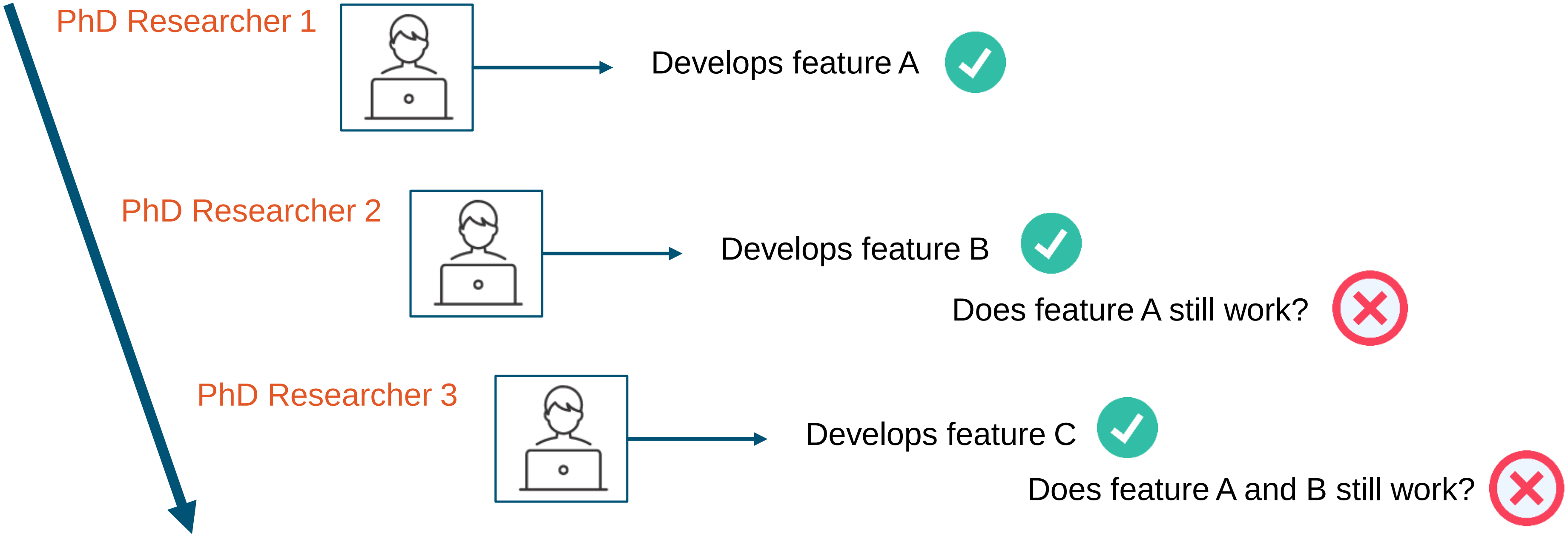
You can add an image to this registry with the following commands:

```
docker build -t git.rz.tu-bs.de:4567/l.hermann/suresoft-test-lucas .
```

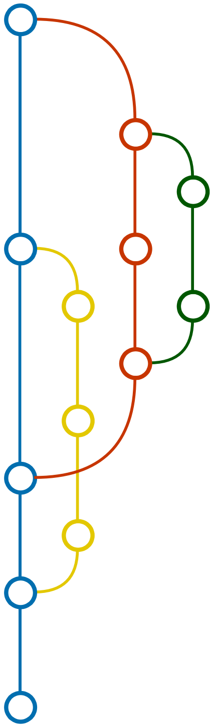
```
docker push git.rz.tu-bs.de:4567/l.hermann/suresoft-test-lucas
```

Part 2: Continuous Integration

Motivation | In Academia



Motivation | Developing in groups



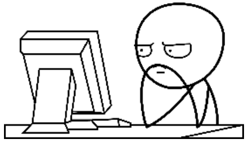
To prevent complex integration:

> “Commit code frequently”
> [Duval et al. practices]

> “Everyone commits to the mainline everyday”
> [Fowler practices]

⇩

Merge conflicts, bugs, defects, broken routines



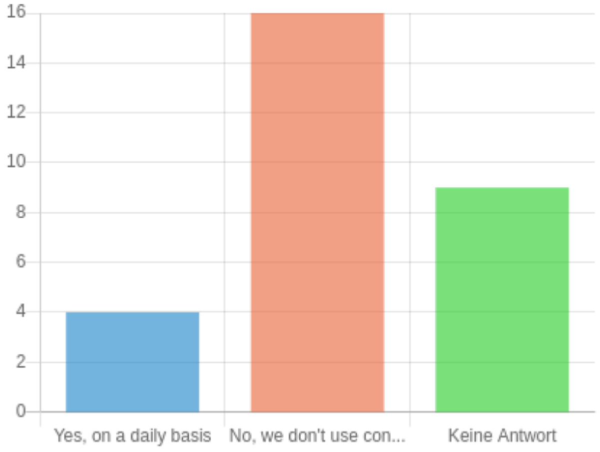
⇩

With CI → Better quality control over new features and their effect on existing implementation – through automated build and test routines

Motivation | Suresoft Survey

Continuous integration not a common practice

Do you use continuous integration?

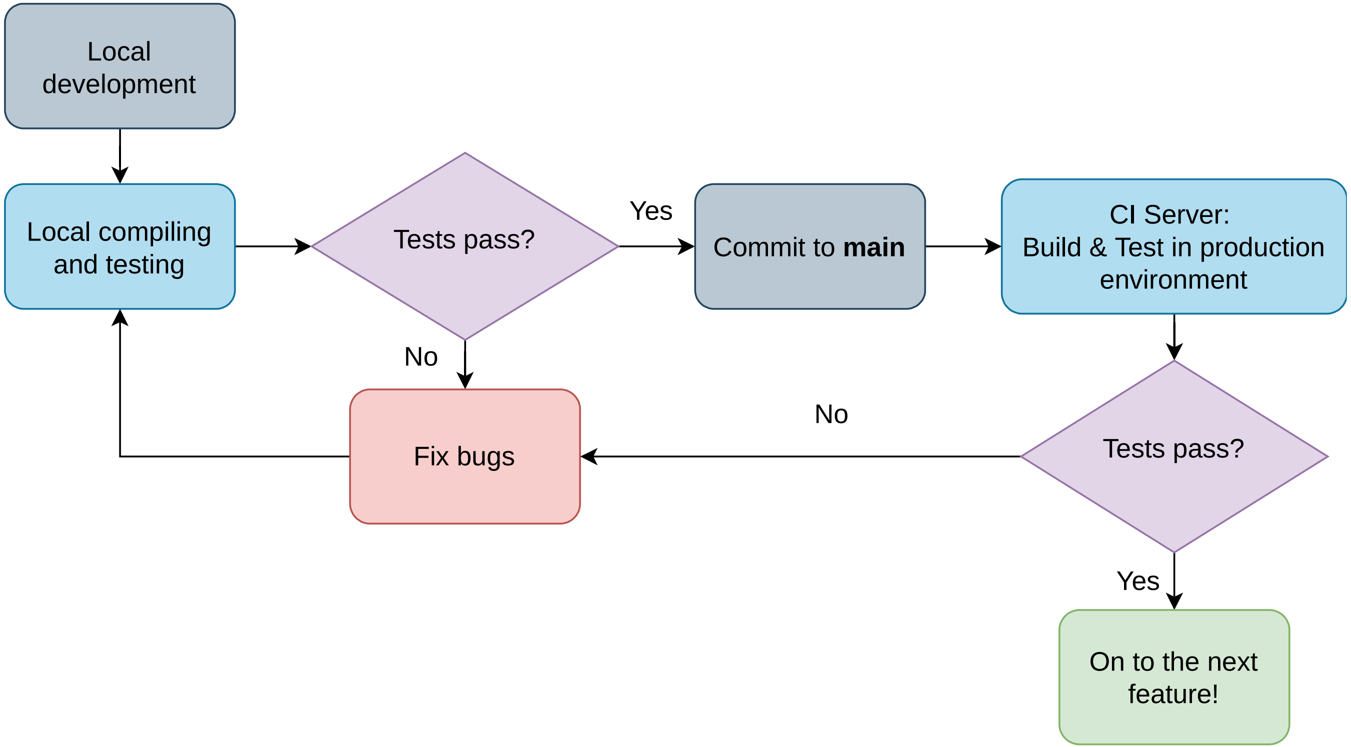


What is continuous integration?

"Practice of automating the integration of code changes from multiple contributors into a single software project."

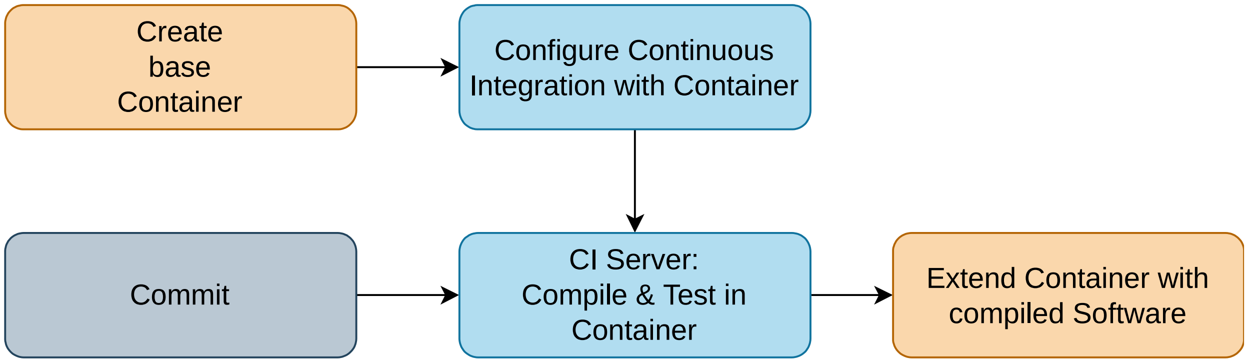
[altassian.com]

Workflow | Continuous integration



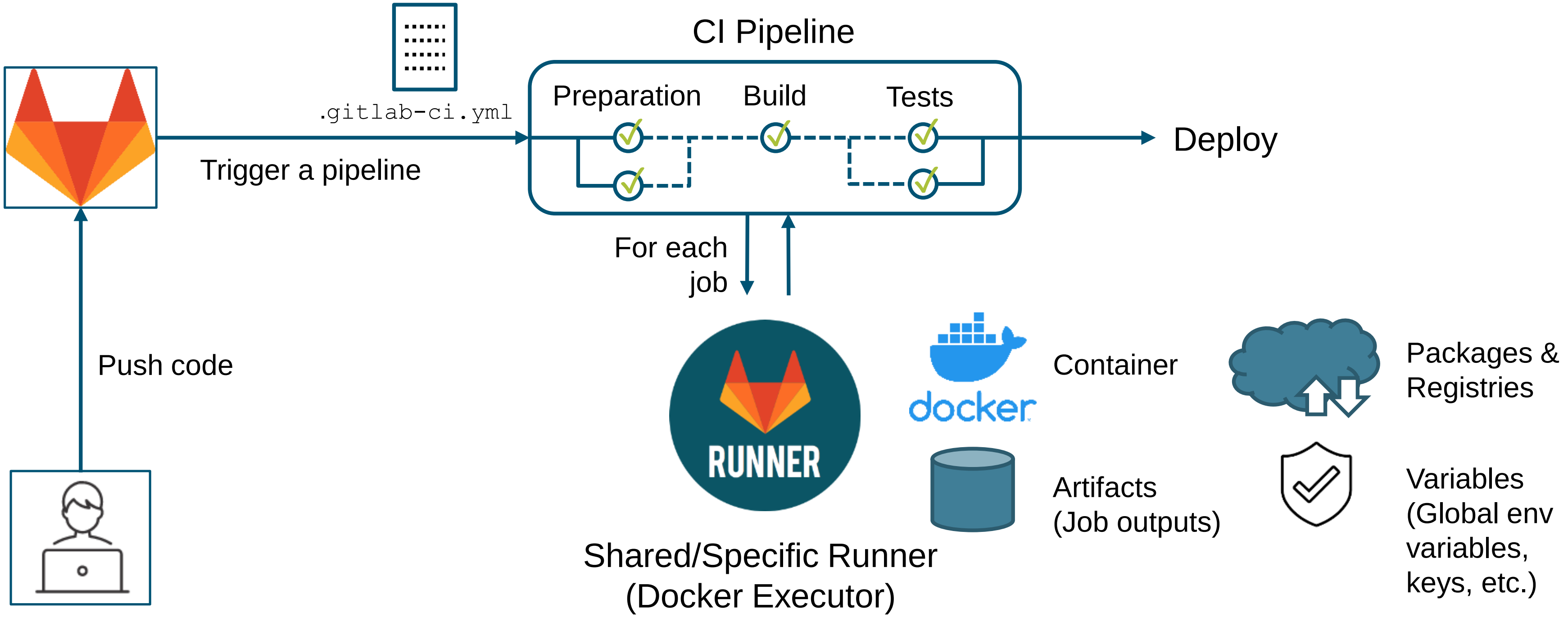
What is Continuous Analysis?

- Combines containerization with the continuous integration approach for reproducibility of research
- Building, testing, deployment and publishing takes places in form of a container (Docker)

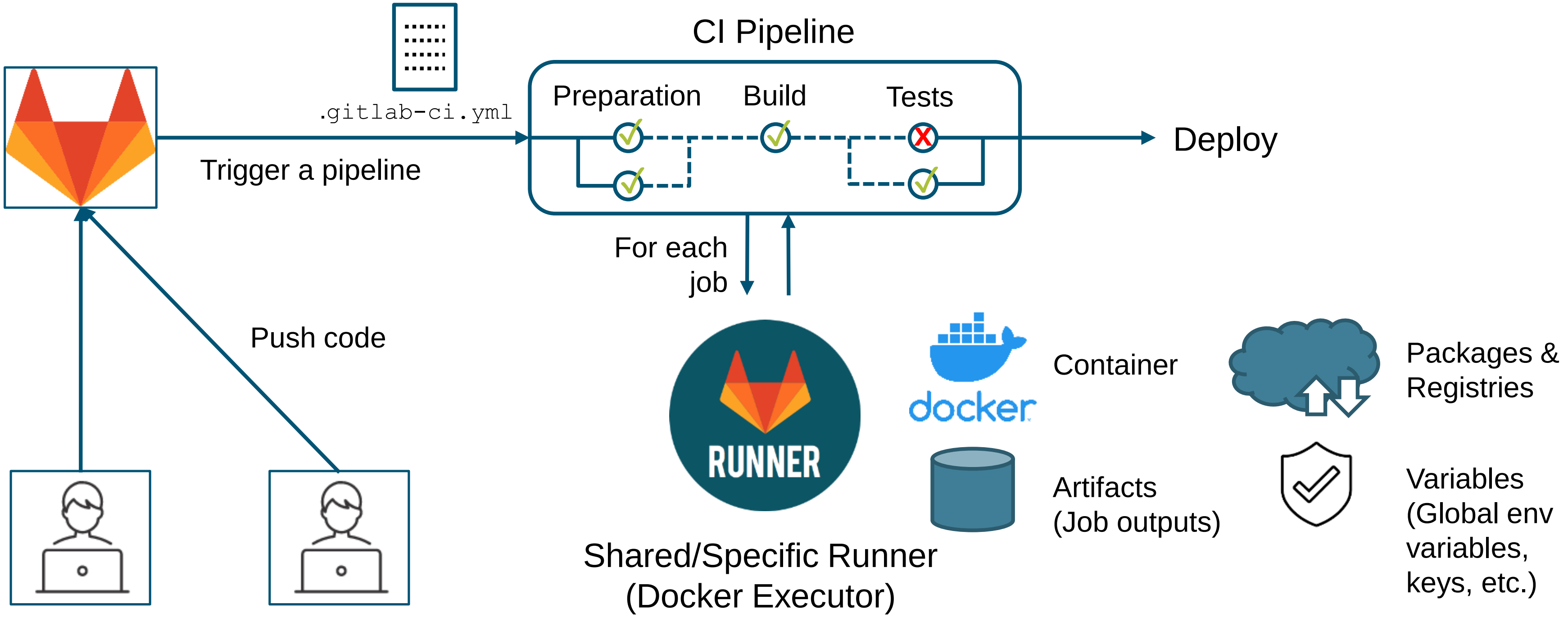


[https://doi.org/10.1038/nbt.3780]

Continuous integration with GitLab



Continuous integration with GitLab



Typical continuous integration pipeline

- Typical CI pipeline incorporates:
  - building the software
  - running extensive test suits
  - providing rapid/continuous feedback to the developers

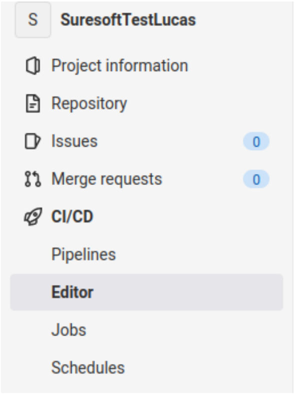
Advantages of CI

- ✓ Increases code-sustainability
- ✓ Increases quality of software development
- ✓ Decrease in repetitive manual process
- ✓ Software is always ready to use
- ✓ Increased robustness of the product
- ✓ Easiness to locate and remove defects
- ✓ Decrease in rate of project failure

[E. Soares, et al.: The Effects of Continuous Integration on Software Development: a Systematic Literature Review. 2021]

Part 2: Hands-on exercises | Create a CI pipeline

Hands-on exercise: CI using GitLab



Part 1: Write the Configuration File

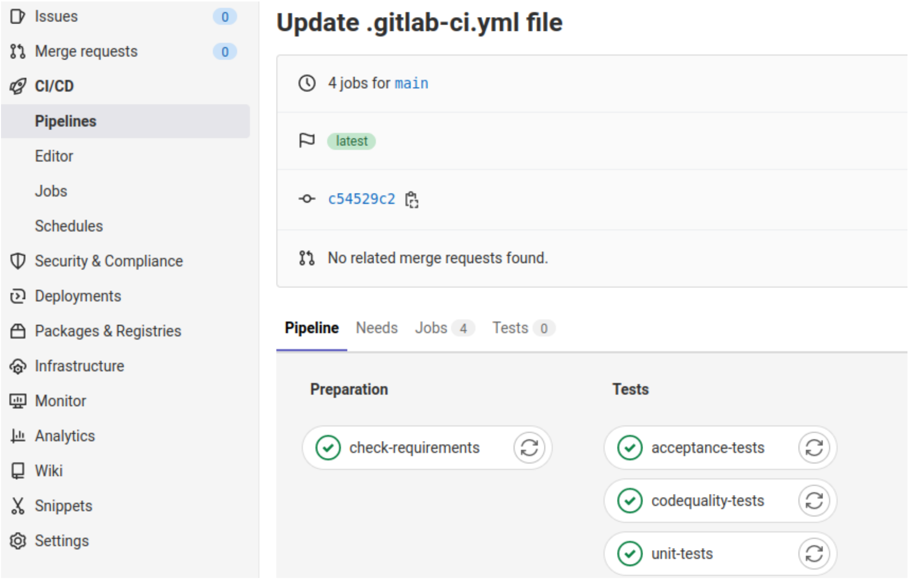
- Navigate to the Pipeline Editor
- Define stages, images and scripts

Browse templates | Help

```yaml
stages:
- preparation
- tests

check-requirements:
  stage: preparation
  image: git.rz.tu-bs.de:4567/hk.sreekumar/ws-ci-matrix-calculator-test-project
  script:
    - python3 -m pytest --version
    - python3 -m paver --version
    - python3 -m pylint --version

unit-tests:
  stage: tests
  image: git.rz.tu-bs.de:4567/hk.sreekumar/ws-ci-matrix-calculator-test-project
  script:
    - python3 -m paver unit_tests

acceptance-tests:
  stage: tests
  image: git.rz.tu-bs.de:4567/hk.sreekumar/ws-ci-matrix-calculator-test-project
  script:
```

Hands-on exercise: CI using GitLab

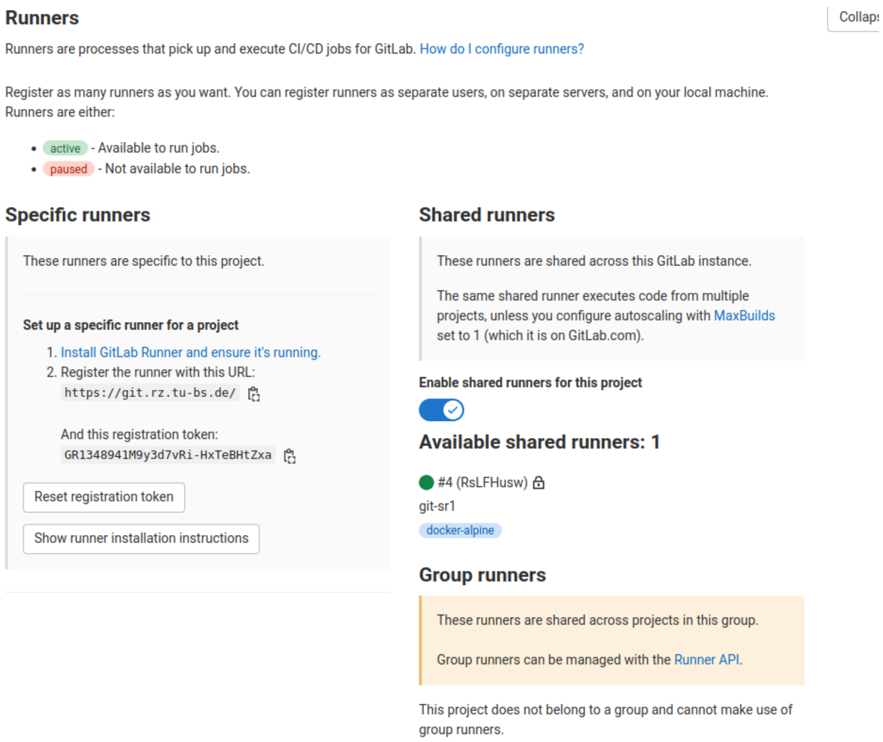
Part 2: Push and check Pipeline status



Hands-on exercise: CI using GitLab

Part 3: GitLab Runners

- Settings, CI/CD, Runners
- GITZ Runner
- Local Runner
- Group Runner on a Server



Hands-on exercise: CI using GitLab

Part 4: Job Artifacts

- In `.gitlab-ci.yml`, use "artifacts" keyword
- Step 1: For unit-tests job, collect "report_unit_tests.html"

```
unit-tests:
  stage: tests
  image: git.rz.tu-bs.de:4567/hk.sreekumar/ws-ci-matrix-calculator-test-project/ubuntu-image
  script:
    - python3 -m paver unit_tests
  artifacts:
    name: reports_unittest
    when: always
    paths:
      - report_unit_tests.html
```

- Step 2: For acceptance-tests job, collect "report_acc_tests.html" and "./data/result.mat"

Hands-on exercise: CI using GitLab

Part 5: Test coverage visualization

- Code coverage is detected with regular expression
- With `pytest` use:

```
unit-tests:
  stage: tests
  image: git.rz.tu-bs.de:4567/hk.sreekumar/ws-ci-matrix-calculator-test-project/ubuntu-image
  script:
    - python3 -m paver unit_tests
  coverage: '/(?i)total.*? (100(?:\.0+)?\%|[1-9]?\d(?:\.\d+)?\%)$/'
```

- Step 1: Collect coverage for both unit-tests and acceptance-tests
- Step 2: Use GitLab-Badges

Best practices

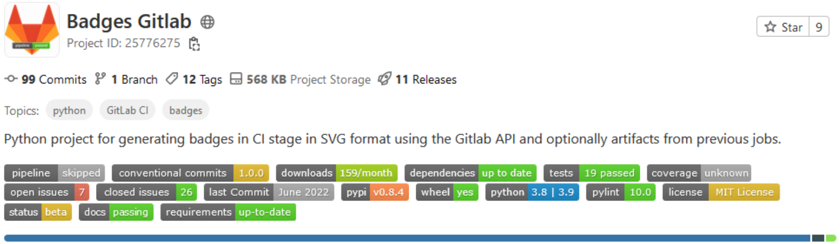
Best practices | Containers

- Use Docker containers during code development
- Share Docker images and Dockerfiles
- Smaller images
  - Use proper base image → start with *alpine* (5 MB) instead of *Ubuntu* (188 MB)
  - Use multistage builds → One image for builds, tests and another for running by copying application artifacts
  - Minimize the number of layers in a docker image
  - Delete unwanted artifacts
- When in CI
  - Assign version your docker image. Example: Git commit hashes
  - Do not store passwords, keys, tokens, etc in images → Supply during runtime (GitLab Variables)

[https://docs.docker.com/develop/dev-best-practices/]

Best practices | CI

- “Commit early and commit often”
- Strive to have successful CI pipelines
- Do not generate the same image every time → Host centrally in a registry for faster pipeline and for saving resources
- Do not hard code sensitive information in .gitlab-ci.yml → Use GitLab Variables (Settings > CI/CD > GitLab Variables)
- Showcase your CI achievements with GitLab Badges (Settings > General > Badges)



[https://about.gitlab.com/topics/ci-cd/continuous-integration-best-practices/]

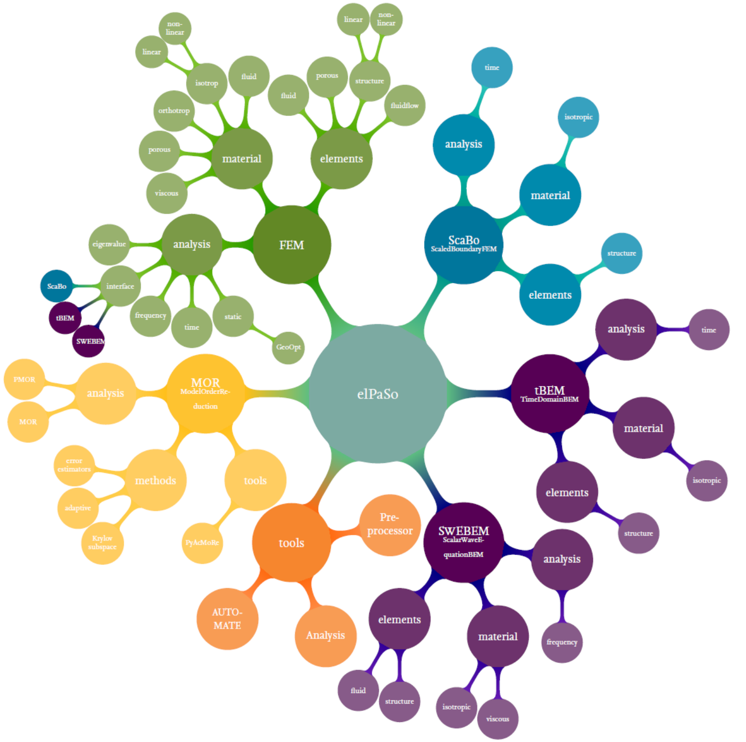
Part 3: Demonstration of elPaSo container approaches and CI pipeline

elPaSo | About

Elementary **Pa**rallel **So**lver (elPaSo)

- Performs vibroacoustic analysis in the modal, static, time and frequency domain
- Based on FEM, BEM, SBFEM
- Efficient computing strategies - parallel computing, model order reduction

tu-bs.de/en/ina/institute/ina-tech/research-code-elpaso



Source: InA/TU Braunschweig

elPaSo | CI Pipeline

Preparing environment ⇨ Software compilation ⇨ Unit and acceptance testing ⇨ Documentation – tutorial and technical ⇨ Release of stable software images

Preparation
- prep-documentation
- prep-gnu-ubuntu
- prep-intel-ubuntu

Build
- build-gnu
- build-intel

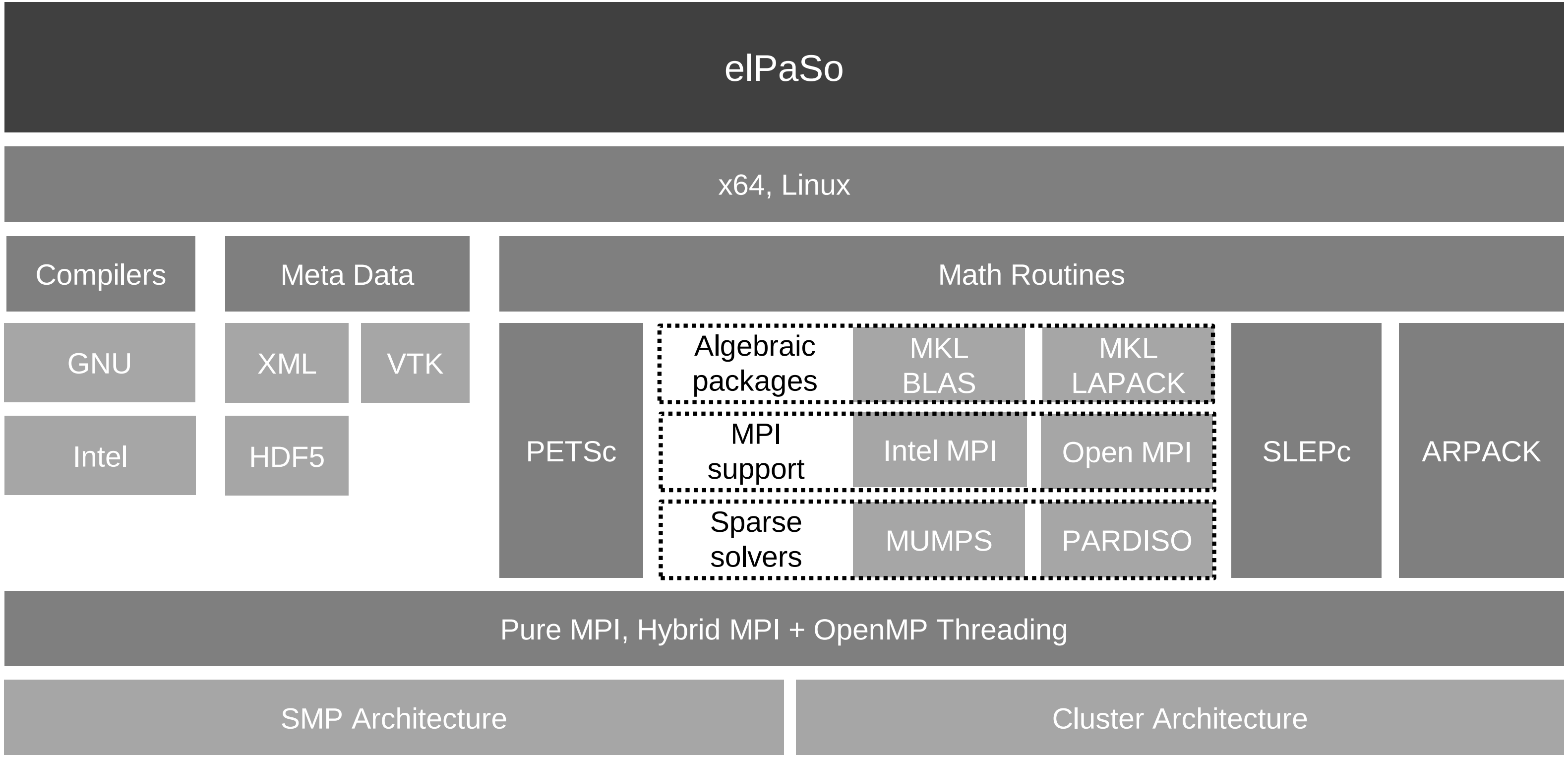
Tests
- tests-integration-gnu
- tests-integration-intel
- tests-lint-clang-tidy-gnu
- tests-lint-clang-tidy-intel
- tests-performance-gnu
- tests-performance-intel
- tests-unit-gnu
- tests-unit-intel

Documentation
- docu-doxygen
- docu-technical
- docu-tutorials

Deploy
- release-docker-gnu
- release-docker-intel

https://git.rz.tu-bs.de/akustik/elPaSo

elPaSo | Software architecture



Source: InA/TU Braunschweig

elPaSo | Containerisation

Primary image : Standard environment (heavy, used by developers and CI)
Software image: Built software with all dependencies (lightweight, used by users)

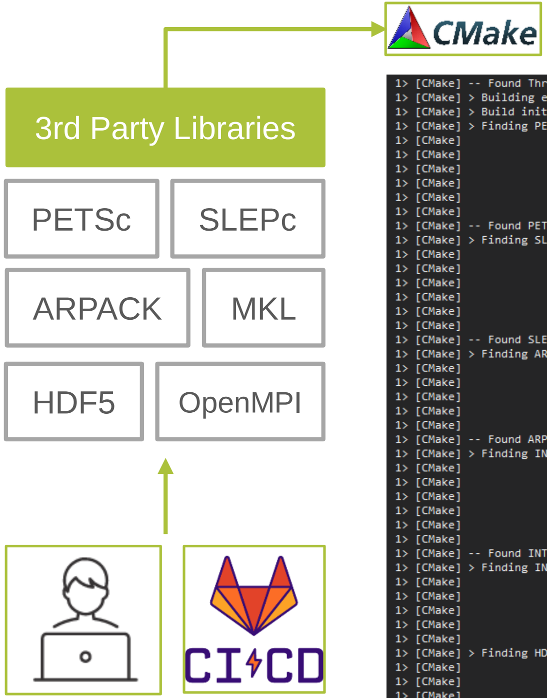
Our platform choice:

- **Docker images**
- **Singularity images** from respective docker images when executing in parallel

Primary images	**Software images (stable releases)**
elpaso-baseimage-ubuntu-x64	elpaso-gnu-ubuntu-x64:$CI_COMMIT_SHA
elpaso-baseimage-intel-ubuntu-x64	elpaso-intel-ubuntu-x64:$CI_COMMIT_SHA

https://git.rz.tu-bs.de/akustik/elPaSo/container_registry

elPaSo | Managing dependencies




```
1> [CMake] -- Found Threads: TRUE
1> [CMake] > Building elPaSo complex module: elpasoC
1> [CMake] > Build initiated for Linux
1> [CMake] > Finding PETSc ...
1> [CMake]              PETSC VERSION       : 3.14.2
1> [CMake]              PETSC ROOT    DIR   : /software/libsI20/petsc-3.14.2/intel-cxx-complex-o
1> [CMake]              PETSC INCLUDE DIR   : /software/libsI20/petsc-3.14.2/intel-cxx-complex-o/include; /softwar
1> [CMake]              PETSC LIBRARY DIR   : /software/libsI20/petsc-3.14.2/intel-cxx-complex-o/lib
1> [CMake]              PETSC LIBS          : /software/libsI20/petsc-3.14.2/intel-cxx-complex-o/lib/libdmumps.a;
1> [CMake]              PETSC COMPILER FLAGS:
1> [CMake] -- Found PETSC: /software/libsI20/petsc-3.14.2/intel-cxx-complex-o/lib/libdmumps.a;/software/libsI20/pet
1> [CMake] > Finding SLEPc ...
1> [CMake]              SLEPC VERSION       : 3.14.1
1> [CMake]              SLEPC ROOT    DIR   : /software/libsI20/slepc-3.14.1/intel-cxx-complex-o
1> [CMake]              SLEPC INCLUDE DIR   : /software/libsI20/slepc-3.14.1/intel-cxx-complex-o/include; /softwar
1> [CMake]              SLEPC LIBRARY DIR   : /software/libsI20/slepc-3.14.1/intel-cxx-complex-o/lib
1> [CMake]              SLEPC LIBS          : /software/libsI20/slepc-3.14.1/intel-cxx-complex-o/lib/libslepc.so
1> [CMake]              SLEPC COMPILER FLAGS:
1> [CMake] -- Found SLEPC: /software/libsI20/slepc-3.14.1/intel-cxx-complex-o/lib/libslepc.so
1> [CMake] > Finding ARPACK ...
1> [CMake]              ARPACK ROOT    DIR   : /software/libsI20/ARPACK
1> [CMake]              ARPACK INCLUDE DIR   : /software/libsI20/ARPACK
1> [CMake]              ARPACK LIBRARY DIR   : /software/libsI20/ARPACK
1> [CMake]              ARPACK LIBS          : /software/libsI20/ARPACK/libarpack_intel-cxx-complex-o.a;/software
1> [CMake]              ARPACK COMPILER FLAGS:
1> [CMake] -- Found ARPACK: /software/libsI20/ARPACK/libarpack_intel-cxx-complex-o.a;/software/libsI20/ARPACK/libpa
1> [CMake] > Finding INTELMPI ...
1> [CMake]              INTELMPI VERSION       : 2019.6.166
1> [CMake]              INTELMPI ROOT    DIR   : /software/intel2020/impi/2019.6.166/intel64
1> [CMake]              INTELMPI INCLUDE DIR   : /software/intel2020/impi/2019.6.166/intel64/include
1> [CMake]              INTELMPI LIBRARY DIR   : /software/intel2020/impi/2019.6.166/intel64/lib;/software/intel20
1> [CMake]              INTELMPI LIBS          : /software/intel2020/impi/2019.6.166/intel64/lib/release/libmpi.so
1> [CMake]              INTELMPI COMPILER FLAGS:
1> [CMake] -- Found INTELMPI: /software/intel2020/impi/2019.6.166/intel64/lib/release/libmpi.so
1> [CMake] > Finding INTELMKL ...
1> [CMake]              INTELMKL ROOT    DIR   : /software/intel2020/mkl
1> [CMake]              INTELMKL INCLUDE DIR   : /software/intel2020/mkl;/software/intel2020/mkl/include
1> [CMake]              INTELMKL LIBRARY DIR   : /software/intel2020/mkl/lib/intel64/
1> [CMake]              INTELMKL LIBS          : /software/intel2020/mkl/lib/intel64/libmkl_intel_lp64.a;/software
1> [CMake]              INTELMKL COMPILER FLAGS:
1> [CMake] > Finding HDF5 ...
1> [CMake]              HDF5 VERSION       : 1.12.0
1> [CMake]              HDF5 ROOT    DIR   : /software/libsI20/hdf5-1.12.0/intel-opt
1> [CMake]              HDF5 INCLUDE DIR   : /software/libsI20/hdf5-1.12.0/intel-opt/include
1> [CMake]              HDF5 LIBRARY DIR   : /software/libsI20/hdf5-1.12.0/intel-opt/lib
1> [CMake]              HDF5 LIBS          : /software/libsI20/hdf5-1.12.0/intel-opt/lib/libhdf5.so
1> [CMake]              HDF5 COMPILER FLAGS:
1> [CMake] -- Found HDF5: /software/libsI20/hdf5-1.12.0/intel-opt/lib/libhdf5.so
```

elPaSo | Managing dependencies

CMake

CONAN registry in GitLab

```
1> [CMake] -- Downloading conan.cmake from https://github.com/conan-io/cmake-conan
1> [CMake] > Conan installation for intel compilers...
1> [CMake] -- Conan: checking conan executable
1> [CMake] -- Conan: Found program /usr/bin/conan
1> [CMake] -- Conan: Version found Conan version 1.39.0
1> [CMake] |
1> [CMake] -- Conan executing: /usr/bin/conan install . --build missing --settings build_type=Release --s
1> [CMake] Configuration:
1> [CMake] [settings]
1> [CMake] arch=x86_64
1> [CMake] arch_build=x86_64
1> [CMake] build_type=Release
1> [CMake] compiler=intel
1> [CMake] compiler.libcxx=libstdc++11
1> [CMake] compiler.version=19.1
1> [CMake] os=Linux
1> [CMake] os_build=Linux
1> [CMake] [options]
1> [CMake] [build_requires]
1> [CMake] [env]
1> [CMake]
1> [CMake] conanfile.txt: Installing package
1> [CMake] Requirements
1> [CMake]     arpack-real/2.1@ina+elpaso/stable from 'gitlab' - Cache
1> [CMake]     hdf5/1.12.0@ina+elpaso/stable from 'gitlab' - Cache
1> [CMake]     petsc-real/3.14.4@ina+elpaso/stable from 'gitlab' - Cache
1> [CMake]     slepc-real/3.14.2@ina+elpaso/stable from 'gitlab' - Cache
1> [CMake] Packages
1> [CMake]     arpack-real/2.1@ina+elpaso/stable:b60d3ded6fd63ebad90ada3d287cdfc4752f66d8 - Cache
1> [CMake]     hdf5/1.12.0@ina+elpaso/stable:b60d3ded6fd63ebad90ada3d287cdfc4752f66d8 - Cache
1> [CMake]     petsc-real/3.14.4@ina+elpaso/stable:b60d3ded6fd63ebad90ada3d287cdfc4752f66d8 - Cache
1> [CMake]     slepc-real/3.14.2@ina+elpaso/stable:b60d3ded6fd63ebad90ada3d287cdfc4752f66d8 - Cache
1> [CMake]
1> [CMake] Installing (downloading, building) binaries...
1> [CMake] arpack-real/2.1@ina+elpaso/stable: Already installed!
1> [CMake] hdf5/1.12.0@ina+elpaso/stable: Already installed!
1> [CMake] petsc-real/3.14.4@ina+elpaso/stable: Already installed!
1> [CMake] slepc-real/3.14.2@ina+elpaso/stable: Already installed!
1> [CMake] conanfile.txt: Generator txt created conanbuildinfo.txt
1> [CMake] conanfile.txt: Generator cmake_find_package created Findpetsc-real.cmake
1> [CMake] conanfile.txt: Generator cmake_find_package created Findslepc-real.cmake
1> [CMake] conanfile.txt: Generator cmake_find_package created Findarpack-real.cmake
1> [CMake] conanfile.txt: Generator cmake_find_package created Findhdf5.cmake
1> [CMake] conanfile.txt: Aggregating env generators
1> [CMake] conanfile.txt: Generated conaninfo.txt
```

openmpi/4.0.0@ina+elpaso/stable
4.0.0 Conan

hdf5/1.12.0@ina+elpaso/stable
1.12.0 Conan

arpack-real/2.1@ina+elpaso/stable
2.1 Conan

arpack-complex/2.1@ina+elpaso/stable
2.1 Conan

slepc-real/3.14.2@ina+elpaso/stable
3.14.2 Conan

slepc-complex/3.14.2@ina+elpaso/stable
3.14.2 Conan

petsc-complex/3.14.4@ina+elpaso/stable
3.14.4 Conan

petsc-real/3.14.4@ina+elpaso/stable
3.14.4 Conan

CONAN.io
C/C++ Package Manager

CI/CD

elPaSo | Managing dependencies

- Easy elPaSo installation
- Time consuming 3rd party library installation can be avoided (1.5 hours → ~1 minute)
- Reduced Docker image size in CI (32 GB → 7 GB [Intel] | 1 GB [GNU])
- Docker script for elPaSo dependencies is now least complicated

Coming up next in workshop series

Suresoft workshop series

Every 4 weeks

1.	Version Control using Git	June 13
2.	Clean Code and Refactoring	July 11
3.	Introduction to Software Testing	August 8
4.	Introduction to Continuous Integration (CI) using GitLab and Containerization	September 5
5.	**Principles of Software Engineering**	**October 10**
6.	Introduction to Design Patterns	November 7
7.	Working with legacy code	TBA
8.	Test Driven Development	TBA
9.	Documentation	TBA

Thank you for your attention