Testing and Validation Framework

Document Control Information

Settings	Value
Document Identifier:	D4.3
Project Title:	ExPaNDS
Work Package:	WP4
Document Author(s):	Jason Brudvik (MAX IV Laboratory), Silvan Schoen (DESY), Zdenek Matej (MAX IV Laboratory), Anton Barty (DESY)
Document Reviewer(s):	Sophie Servan (DESY), Krisztian Pozsa (PSI)
Responsible Partner:	MAX IV Laboratory, Lund University
Doc. Issue:	1.0
Dissemination level:	Public
Date:	30/11/2021

Abstract

This document presents a framework for testing and validating ExPaNDS services against reference data sets. The process is demonstrated for the case of Jupyter notebook type services.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 857641.

1. Executive Summary

Within scientific communities, there is a wide range of scientific software for specific use cases. Software used for data analysis pipelines in established and accepted research workflows includes command-line processing, interactive graphical desktop applications, single-CPU applications and MPI-based multi-node software running on high-performance computing (HPC) infrastructure. It often requires access to specific hardware or computing accelerator resources. The purpose of a testing and validation framework is to verify that software is available and operating as intended on different research infrastructures. Testing should be ideally performed in an automated manner. This provides a path for delivering analysis software as a service on shared infrastructure, including but not limited to European Open Science Cloud (EOSC) and shared HPC resources.

For testing software deployments, ExPaNDS partners have defined a set of nine reference science cases and related data sets covering several Photon and Neutron (PaN) experimental techniques, representing real-world data analysis workflows. The reference cases consist of available experimental data sets, existing software and scientific workflows from different ExPaNDS sites. The goal of this work is to establish a framework that assures that data analysis services can be validated against reference data sets.

In this document, we describe a framework for testing and deploying software that is made available for users across research facilities with heterogeneous compute infrastructures. A general testing pipeline is described first and then we focus on a particular case of validating Jupyter notebook services as a concrete example. The framework described herein serves to make sure that the services are working correctly as different prototypes are adapted by various partners.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 857641.

Table of Contents

Executive Summary **1**

Background **4**
- ExPaNDS context of the testing framework 4
- EOSC Synergy service quality assurance 4
- Quality assurance for ExPaNDS services 5

General software testing pipeline **5**
- Forms of scientific software installations 6
  - Personal user installations 6
  - Facility installations 6
  - Delivery of software through containerization 6
- A CI/CD pipeline for application image deployment 7
  - CI building stage 7
  - CI testing stage 8

A Specific example for Jupyter-notebooks **8**
- Jupyter-notebooks 8
- Jupyter kernels 9
- Jupyter installations 9
  - Docker and Kubernetes JupyterHub installations 9
  - HPC JupyterHub installations 9
- Testing and validation of jupyter-notebooks services 10
  - Tools for testing and validation 10
  - jnbv 10
  - jupyter-notebook-validation 10
  - How to test and validate 11
- Continuous integration pipeline 12
  - Reasons to trigger pipeline 12
  - Pipeline steps 12
  - Docker and Kubernetes pipeline 12
  - HPC pipeline 12
  - Gitlab pipeline output 13
  - Status of CI pipeline 13

Summary **13**

Acknowledgements **14**

References **14**

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 857641.

1. Background

1.1. ExPaNDS context of the testing framework

ExPaNDS project strategy is defined in the following documents:

- [1)] "ExPaNDS General Architecture description in relation to the EOSC services"
- [2)] "Guidelines for implementing the national Photon and Neutron RI's analysis services within the EOSC"
- [3)] "Photon and Neutron reference data sets"

The national photon and neutron (PaN) research infrastructures (RI's) participating in ExPaNDS operate a heterogeneous set of data storage and computing infrastructures[1)]. While scientific workflows often share common software elements and facilities generally share common scientific data management practices[2)], the operating environment at each facility is slightly different for largely historical reasons. Moving towards horizontal infrastructures, therefore, requires an automated testing and validation framework to verify that shared scientific software can execute properly at other facilities and ensure long term reliability and sustainability of the service. In ExPaNDS, we develop this infrastructure by focussing on sharing software required for the reference data sets and by developing services and related tools that are interoperable between ExPaNDS partners.

ExPaNDS partners have defined a set of 9 reference science cases and related data sets[3)] covering several PaN experimental techniques representing real-world data analysis workflows. The reference cases consist of available experimental data sets, existing software and scientific workflows available at different ExPaNDS RI's. The aim of ExPaNDS work package 4 (WP4) is to make the data analysis workflows related to the reference data sets available as European Open Science Cloud (EOSC) services. This should improve the availability of the experimental methods for a wider user community, ensure repeatability of scientific results with open data and increase PaN community user experience by providing similar service frontends for users at different RI's. The purpose of the ExPaNDS testing framework is to validate that software and services required for executing analysis workflows function as intended at the partners' facilities.

1.2. EOSC Synergy service quality assurance

Concerning the broader EOSC environment, ExPaNDS project appreciates that a partner project "EOSC Synergy" has developed baseline criteria for the software management lifecycle of EOSC services, including data repositories, and a concept of automated validation process featuring continuous integration (CI) and continuous delivery (CD)

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 857641.

pipelines. Two outcomes of the EOSC Synergy project are documents describing these criteria and guidelines for software and services:

- [4] "A set of Common Software Quality Assurance Baseline Criteria for Research Projects"
- [5] "A set of Common Service Quality Assurance Baseline Criteria for Research Projects"

We draw on these reference documents in developing the ExPaNDS testing framework. It seems feasible that also software developed within the ExPaNDS project can follow the EOSC Synergy guidelines described in the document defining baseline criteria for software[4].

1.3. Quality assurance for ExPaNDS services

We note that general aspects of service validation for web services, web applications and platforms which are compositions of multiple services and web applications are already covered by EOSC Synergy in the second document "A set of Common Service Quality Assurance Baseline Criteria for Research Projects"[5]. That document describes the various performance, security testing, service monitoring, automated deployment, requirements on documentation, support and policies, among other topics. Additionally, once a service is onboarded in the EOSC, it will be monitored according to the EOSC-hub specifications[6] in EOSC's ARGO service[7]. And so we do not repeat these topics here as they are already adequately covered elsewhere.
On the other hand, the testing framework introduced here is understood as a set of good practice guidelines, reference implementations and original software. The last is developed only when critical tools are not already available.

2. General software testing pipeline

ExPaNDS testing and validation framework introduced in the next paragraphs aims to present a supporting structure to make sure that ExPanDS analysis services are working correctly as the different prototypes are adapted by the various partners. It consists of

- general guidelines for testing and validating scientific data analysis workflows,
- references to related available tools, testing, validation and monitoring services,
- a reference example implementation of a continuous validation pipeline.

The framework has to deal with a wide range of software that exists in the scientific community. It includes software used for data analysis pipelines in established and accepted research workflows. This software is highly varied, ranging from command-line processing to interactive graphical desktop applications, from single-CPU applications to MPI-based multi-node software running on high-performance computing (HPC) infrastructure, sometimes requiring access to specific hardware or GPU resources. Nevertheless, the same

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 857641.

premises and challenges apply across platforms. Later in this document, we describe the case of Jupyter notebooks in more specific detail by way of an example for a specific class of application.

In general, the objectives are to:

- Make software readily available to a large scientific audience
- Have the software checked and validated for a wide array of arbitrary hardware and operating systems; and
- Encourage analyses within publications be fully replicable

The purpose of a testing and validation pipeline is to automate the process of delivery of scientific software, and ideally provide a path to analysis as a service on shared infrastructure (including but not limited to EOSC and shared HPC resources). Open reference datasets[3)] should provide complex input for this validation process.

2.1. Forms of scientific software installations

2.1.1. Personal user installations

Research software is often developed as open-source projects by the community or within certain research groups. In many cases distribution of the software is voluntary or performed alongside the primary function of research, leading to an absence of maintained installation packages and dependencies that are in practice limited to distinct operating systems or hardware. Maintaining software distribution packages for multiple environments implies an ongoing effort over years and is often not within the scope envisaged by the developers who are under pressure to write the next paper rather than support existing software.

This model of software distribution leads to challenges for the user scientist. Research software must often be installed personally, frequently with no other option than from source and possibly without even having root privileges. For the scientist, installing sometimes incompatible software is a major source of frustration and wasted effort. Being able to access and use validated packages would be a major advantage.

2.1.2. Facility installations

Some facilities employ staff to install software for the user on infrastructure maintained by the facility. Here, installation does not pose a problem for the user, however, control over the specific software or even software version might not be possible. Long term sustainability of such service also depends on human resources available at different RI's and reproducibility of the service on different facilities may not be a simple issue.

2.1.3. Delivery of software through containerization

Delivery of containerized software may present an invaluable tool to meet the defined goals of distributing validated research software. Application distribution in a containerized fashion can address the challenges mentioned above. In particular:

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 857641.

- With containers, individual user installation of the software from source or packages becomes unnecessary.
- It is possible to unambiguously track which container with which software version was used in a specific use case.
- Additionally, containerized software can be deployed on a wide range of operating systems.

However, even though the user doesn't have to install the scientific software itself, it is still necessary to provide software to execute containerized applications on personal systems and RI's computing infrastructure. If facility resources are used, the availability of containerization software may differ between RI's as the user rights management is not trivial. Concerning the choice of container infrastructure, we follow the options of Singularity and Docker for their distinctive advantages and therefore focus on these technologies.

2.2. A CI/CD pipeline for application image deployment

Containerized application distribution can be combined with modern CI/CD infrastructure to perform automatic software tests before the software containers are distributed. We chose to use GitLab for our initial containerized applications. GitLab offers not only a container image registry but also structures to define CI/CD pipelines for the container images. This enables both software testing and container deployment in an existing and widely adopted environment.

In short, we will adopt the use of modern CI/CD software development protocols for scientific software distribution and analysis workflows identified for ExPaNDS cases, based on GitLab as an existing, tested and well-accepted platform.

Validating software services for ExPaNDS reference use cases involves two steps:

1. CI building stage
2. CI testing stage

Appropriate CI/CD pipelines can be implemented with the aid of DESY EOSC PaN GitLab service available at https://eosc-pan-git.desy.de.

2.2.1. CI building stage

In the first step, whenever a new version of the software is available, a new image can be built in a GitLab pipeline and pushed to its registry (mirroring the images on an alternative registry is also possible). For the build process, a Docker in Docker solution can be used. A Docker image is loaded which itself has Docker installed in it. With this approach, a containerized application image is built from the provided Dockerfile, that contains the installation from source instructions of the application. Subsequently, the image is pushed to the local GitLab registry. Importantly, the software is only released publicly once it has passed the subsequent functionality testing.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 857641.

2.2.2. CI testing stage

In the second stage, the pre-built image is tested for functionality. The goal is not to test the software itself, which is expected to be fully operational, but the functionality of the software which is run within a container. It is verified that the software runs as expected and does not produce errors on a standard test data set, but not to question or validate the scientific results themselves. Thus, only End-to-End tests are applicable, which implies that only a very limited number of input parameters and outputs can be verified. Here, the reference datasets will be very valuable candidates for providing the inputs of the tests.
The approach for the testing stage is similar to the CI building stage. A Docker in Docker image can be used to start the containerized application. Arbitrary predefined tests can now be performed to ensure the functionality of the application within the container.

Beyond the general level described above, details of each CI/CD pipeline will have to be developed on a case by case basis depending on the particular software and test data set.

3. A Specific example for Jupyter-notebooks

We focused on Jupyter notebook workflows to pursue a concrete example for developing a testing framework. Jupyter notebooks are an invaluable tool for the reference data sets for both analysis and data visualization. The jupyter-notebook services were already adopted on several ExPaNDS sites and it was natural to start implementing common scientific cases with them. Additionally, this type of service is new to some PaN facilities, PaN RI's are developing extensions and tools for jupyter-notebooks services, which also implies requirements on their testing and validation. And so, whereas the previous section describes a general testing workflow for validating scientific software deployments using CI/CD pipelines and reference data sets[3], the following section introduces in detail a specific example of jupyter-notebook service validation against reference data sets and related tools.

3.1. Jupyter-notebooks

Jupyter-notebooks are a useful way of providing a data analysis service, allowing researchers simple means for accessing and analyzing experimental data.

To allow researchers to focus on data analysis, the infrastructure and analysis tools have been set up for them as a service which:

- Is accessible via any web browser
- Gives researchers access to all their data
- Has all computing infrastructure setup
- Has all analysis tools pre-installed

Different experiments and analysis techniques require various analysis tools and computing resources, and these need to be customized for different research interests.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 857641.

3.2. Jupyter kernels

The software environments providing interactive programming in the Jupyter notebooks are referred to as Jupyter kernels.

These kernels are built upon python environments often created with conda, which is a package, dependency and environment management system. On HPC clusters, these environments can also be created with the “lmod” environment module system[8)].

3.3. Jupyter installations

Using Jupyter notebooks in a multi-tenant environment is tricky, fortunately, JupyterHub solves the problems. It provides connections to the authorization of users, shared data storage and shared computing resources.

There are different sorts of JupyterHub installations, some of which will be discussed here.

3.3.1. Docker and Kubernetes JupyterHub installations

One method of setting up a JupyterHub installation is to use docker images.

When a user connects to the service, a docker image is selected, and then a new docker container is started up from this image just for that user, completely separated from all other users.

The docker images contain all the necessary system tools and Jupyter kernels needed. The self-contained nature of docker images means that these images are easily reused on different computing infrastructures and at different institutions.

There are various options for the orchestration framework used for handling the operation of the docker containers, user authentication, data access, and compute resource allocation. Two of the popular options are Docker Swarm and Kubernetes, both of which have been used in the work for this work package.

3.3.2. HPC JupyterHub installations

With the installation of JupyterHub on an HPC cluster, docker images are generally unnecessary as much of the necessary software and system tools have already been installed and made available for research analysis.

The HPC system has the Jupyter kernels preinstalled. When a user connects to JupyterHub running on an HPC, a batch job is started on a node using a job scheduler, such as SLURM, as was used in the work for this work package.

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 857641.

3.4. Testing and validation of jupyter-notebooks services

With all the various components involved in a JupyterHub installation being dependent upon one another, it is important to have validation checks so that everything is kept in working order and that analysis results on a specific dataset can be reproduced a week, a month, or a year from now.

The work for this work package focused on Jupyter kernels – checking that:

- The Jupyter kernel can be rebuilt with the necessary software modules
- The analysis routines can be executed without errors
- The results of the execution match what was obtained originally

3.4.1. Tools for testing and validation

Two collections of software were produced for this work package, and are publicly available:

- jnbv
  - https://gitlab.com/MAXIV-SCISW/JUPYTERHUB/jnbv
- jupyter-notebook-validation
  - https://gitlab.com/MAXIV-SCISW/JUPYTERHUB/jupyter-notebook-validation

3.4.2. jnbv

jnbv is a python module for testing Jupyter kernel and Jupyter notebooks against each other.

With this module, it is possible to:

1. Execute notebooks in a terminal
2. Execute notebooks non-interactively in a CI pipeline
3. Check execution output for errors
4. Compare the execution output to a known reference output
5. Choose different jupyter kernels to use
6. Log results of notebook executions and tests

jnbv follows EOSC Synergy software quality assurance guidelines[4].

jnbv is used here for validating reference Jupyter notebooks and associated data sets versus Jupyter kernels and vice versa.

3.4.3. jupyter-notebook-validation

jupyter-notebook-validation is a repository (set of tools, example reference notebooks and configuration files) meant to make use of the python module jnbv for automated validation of production-ready Jupyter kernels.

The software contained in the jupyter-notebook-validation repository is not strictly needed to perform a test of Jupyter kernel and notebook compatibility. It encapsulates tools for configuring an automated continuous validation process for Jupyter notebook based services

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 857641.

providing a complex set of Jupyter kernels for different scientific use cases. In particular, it contains:

- Around 50 Jupyter notebooks, that are used for testing particular Jupyter kernels in use at MAX IV Laboratory and other institutions.
- A Makefile which simplifies the execution of the validation tests via docker images and slurm batch jobs
- example CI/CD configuration for the validation pipeline

This repository has been set up to run as part of the GitLab CI at MAX IV Laboratory for use with several installations of JupyterHub:

- Docker Swarm
- Kubernetes
- HPC

The setup is general enough that it could be easily modified to work at other institutions.

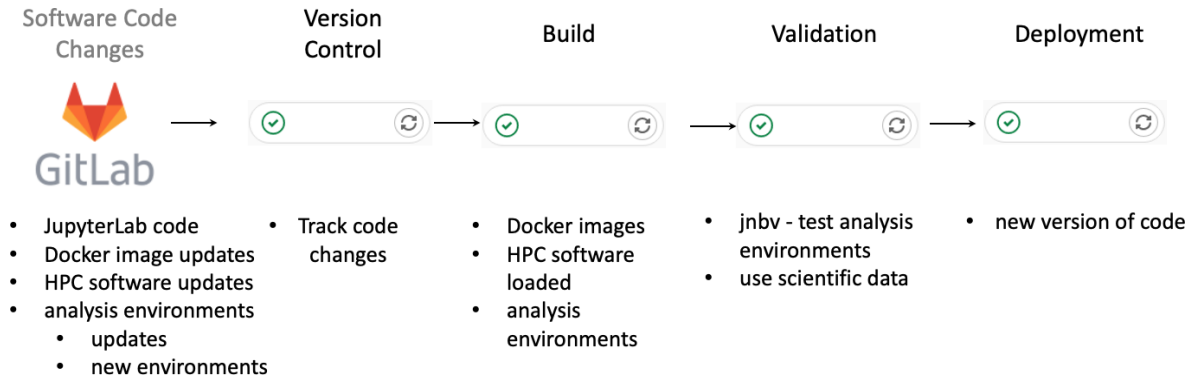

Figure 1: An example jupyter-notebook service quality assurance pipeline.

3.4.4. How to test and validate

The python module jnbv is installed and then used to:

- Execute Jupyter notebooks, which:
  - Load the necessary software
  - Use reference data sets
  - Use short data analysis routines
  - Produce as output a new Jupyter notebook
- Read the execution output
- Test the execution output for errors
- Compare the execution output with the original Jupyter notebook
  - Checking that analysis calculations are the same
  - Any images produced are the same
- Save testing and validation output
  - Log files of test results
  - Jupyter notebooks from executions

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 857641.

3.5. Continuous integration pipeline

As there can be any number of combinations of infrastructures and data analysis tools, an automated continuous integration (CI) validation pipeline is very useful.

This allows for the Jupyter kernels to be automatically tested and verified without any interaction from system maintainers.

The simplified schema of a validation pipeline for Jupyter-notebook service is depicted in Figure 1.

3.5.1. Reasons to trigger pipeline

There are several common reasons why a CI pipeline would be triggered to run:

- Change in software dependency
- Change in Docker Image or HPC system
- Addition of new data analysis routines or kernels

3.5.2. Pipeline steps

The pipeline steps for any installation of JupyterHub are generally the same, but there are some differences. An outline of these steps is presented below.

3.5.3. Docker and Kubernetes pipeline

When the CI pipeline is triggered, say for example a new version of the h5py Python module is installed by modifying the container or conda environment recipe in the git repository, then the CI is set up to:

- Rebuild all docker images using the container or conda environment recipe
- Recreate the dependent Jupyter kernels
- Push docker images to a docker registry
- Pull docker images from docker registry to JupyterHub server
- Run routines to test & validate kernels
- Deploy the new docker images if testing and verification were passed
- Transfer the testing and validation output files to a central storage system for later review

3.5.4. HPC pipeline

For an HPC system the steps are similar, with the notable exception of no docker images being built:

- Recreate the Jupyter kernels relying on this module
- Run routines to test & validate kernels
- Deploy the new kernel if testing and verification were passed
- Transfer the testing and validation output files to a central storage system for later review

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 857641.

3.5.5. Gitlab pipeline output

For this work, a GitLab CI was used (Figure 1), but there are a number of such tools available and anything similar would be able to function the same.

When using GitLab, a record of the CI output is kept and is easily available for review or re-execution if needed, for example in case of temporary network interruptions.

3.5.6. Status of CI pipeline

The CI pipeline described here has been in use for several months in multiple production JupyterHub installations at MAX IV Laboratory and has been tested at ESRF. These installations are orchestrated by:

- Docker Swarm
- Kubernetes
- HPC (slurm)

Which make use of several tools:

- docker
- lmod
- GitLab

And depend upon the testing and validation tools developed for this work package:

- jnbv
- jupyter-notebook-validation

4. Summary

We have outlined tools for testing and validation to be used for ExPaNDS services. These have been developed and delivered into production within the first part of the ExPaNDS project.

A general GitLab CI/CD pipeline for horizontal infrastructures is available within DESY EOSC PaN-cloud services:

- EOSC PaN GitLab: https://eosc-pan-git.desy.de

The testing & validation framework for Jupyter notebooks is set up and in use at MAX IV Laboratory and could be easily configured to work at other ExPaNDS partners' infrastructures:

- The code repositories are publicly available:
  - https://gitlab.com/MAXIV-SCISW/JUPYTERHUB/jnbv
  - https://gitlab.com/MAXIV-SCISW/JUPYTERHUB/jupyter-notebook-validation
- As are the docker images mentioned and the code used to produce them:
  - https://gitlab.com/MAXIV-SCISW/JUPYTERHUB/jupyter-docker-stacks/container_registry

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 857641.

    - https://gitlab.com/MAXIV-SCISW/JUPYTERHUB/jupyter-docker-stacks
- The HPC setup used is also available:
    - https://gitlab.com/MAXIV-SCISW/JUPYTERHUB/jupyterhub-hpc

Acknowledgements

ExPaNDS would like to kindly acknowledge EOSC Synergy for their activities in the development of testing and validation guidelines and tools for EOSC software and services, and in particular, Isabel Campos and her coworkers for the fruitful discussion around the validation and testing of the ExPaNDS services. We would like to also acknowledge Loic Huder (ESRF) for his valuable contribution to jnbv software.

References

1. D. Scardaci, D. Salvat, P. Fuhrmann, A. Barty, A. Ashton, S. Servan: *ExPaNDS General Architecture description in relation to the EOSC services*, zeneodo (2020), doi: 10.5281/zenodo.3697704
2. A. Barty, A. Ashton, P. Fuhrmann, U. Konrad, F. Lang, A. Manzi, Z. Matej, B. Matthews, M. Ounsy, K. Pozsa, C. Reynolds, D. Salvat, S. Servan: *Guidelines for implementing the national Photon and Neutron RI's analysis services within the EOSC*, zenodo (2021), doi: 10.5281/zenodo.4569421
3. A. Ashton, A. Barty, P. Fuhrmann, U. Konrad, F. Lang, Z. Matej, M. Ounsy, C. Reynolds, S. Servan: Photon and Neutron reference data sets: zenodo (2021), doi: 10.5281/zenodo.4558708
4. P. Orviz, G.A. López, D.C. Duma, G. Donvito, M. David, J. Gomes: *A set of Common Software Quality Assurance Baseline Criteria for Research Projects*, Digital CSIC (2020), v3.2, doi: 10.20350/digitalCSIC/12543
5. P. Orviz, M. David, J. Gomes, J. Pina, S. Bernardo, I. Campos, G. Moltó, M. Caballer: *A set of Common Service Quality Assurance Baseline Criteria for Research Projects*, Digital CSIC (2020), doi: 10.20350/digitalCSIC/12533
6. T. Zamani, K. Koumantaros, P. Weber, D. Scardaci, J. Jensen, C. Condurache: *EOSC Hub Technical Specification - Federation Services - Monitoring*, EOSC-hub (2020), link: wiki.eosc-hub.eu/display/EOSCDOC/Monitoring
7. A. Andronidis, M. Tsantekidis, D. Vrcic: *EOSC's ARGO service*, EOSC-portal (last visited Nov, 2021), link: argoeu.github.io, argo.eosc-portal.eu
8. R. McLay, K.W. Schulz, W.L. Barth, T. Minyard: *Best practices for the deployment and management of production HPC clusters*, State of the Practice Reports, SC11 (2011), p. 9, doi: 10.1145/2063348.2063360

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 857641.