D5.1 - Setup of common development platform including CI/CD

Document Properties

Contract Number	101177590
Contractual Deadline	M03 (August 31, 2025)
Dissemination Level	Public
Nature	Report
Author(s)	Hannes Tröpgen (TUD), Jayesh Badwaik (FZJ), Sonja Happ (PARTEC), Mohsen Seyedkazemi Ardebili (UNIBO)
Contributor(s)	Hans-Christian Hoppe (FZJ)
Reviewers	Pierrick Pochelu (LXP), Ujjwal Sinha (FZJ)
Date	August 29, 2025
Keywords	SEANERGYS, HPC, Exascale, Software, Energy Efficiency
Status	Final
Release	1.0

The SEANERGYS project receives funding from the European High Performance Computing Joint Undertaking (JU) under grant agreement no 101177590. The JU receives support from the European Union's Horizon Europe research and innovation programme and Czechia, France, Germany, Greece, Italy, and Spain. Views and opinions expressed are those of the author(s) only and do not necessarily reflect those of the European Union or of the granting authority. Neither the European Union nor the granting authority can be held responsible for them.

Document Status Sheet

Version	Date	Author, Organization	Comment
1.0	August 29, 2025	Hannes Tröpgen (TUD), Jayesh Badwaik (FZJ), Sonja Happ (PARTEC), Mohsen Seyedkazemi Ardebili (UNIBO)	Final Version submitted to EuroHPC JU

Table of Contents

List of Figures

Executive Summary

This deliverable documents the initial setup of the software development platform of the Software for Efficient and Energy-Aware Supercomputers (SEANERGYS) project. It covers the GitLab instance, repositories, access control, issue tracking, basic Continuous Integration and Continous Deployment (CI/CD) infrastructure, shared infrastructure components, and initial High Performance Computing (HPC) integration considerations.

The final software development platform (Gitlab Premium) is being procured and deployed. At the time of writing, the SEANERGYS partners use existing GitLab instances at Jülich Supercomputing Centre (JSC), IT4Innovations (IT4I) at VSB - TUO (VSB) and other partner institutes. The final development platform extends the planning utilities available in the free edition by adding (among other features) push rules, code quality reports and quality analytics, and GitLab epics as work items to create planning hierarchies.

1 Introduction

This document presents the initial configuration and rationale for the SEANERGYS development platform, laying the groundwork for collaborative development and automation across the project consortium.

DevOps platforms play an essential role in modern software projects, as they bring together source code management, automation of testing and deployment, and collaboration features in a single environment. For SEANERGYS, GitLab (in its Premium version) is used as the central DevOps environment because it supports self-hosting, extensibility, and integration with institutional HPC resources, as further detailed in Chapter 2. These capabilities align closely with the project's requirements and provide a foundation for advanced CI/CD workflows, federated collaboration, and integration with external services and HPC systems.

The characteristic features of SEANERGYS is the extension of CI/CD beyond conventional development processes:

1. The project is designed to collaborate between distributed teams and partner institutes.
2. It targets HPC platforms – an area where CI/CD adoption remains technically challenging [1].
3. It integrates DevOps, DataOps, and Machine Learning Operations (MLOps) practices to automate not only code delivery, but also data processing and the full lifecycle of machine learning models.

This holistic approach ensures that SEANERGYS can deliver reproducible and scalable workflows tailored to the demands of energy-efficient Exascale computing. The initial setup documented here represents the first step in this evolution, which will expand to include advanced workflow automation, Machine Learning (ML) model lifecycle management, and large-scale data processing.

After introducing the rationale for using GitLab as the central DevOps environment and describing the repository structure and access policies in Chapter 2, this deliverable details the first implementation of CI/CD pipelines in Chapter 3. Furthermore, it documents the configuration of shared infrastructure components and the integration pathways towards HPC systems, which are critical for the project's computational workloads. Chapter 4 explains the available planning and tracking features of GitLab Premium, and shows how they can be used in SEANERGYS to trace the development from requirements and architecture in the planning phase to code changes and finally software releases. To assess both SEANERGYS' software components and infrastructure, Chapter 5 sketches how metrics interact with GitLab as the central DevOps platform. Finally, Chapter 6 presents a full scenario within GitLab, highlighting its CI/CD capabilities, before the efforts are summarised and next steps described in Chapter 7.

The specific originality of SEANERGYS lies in how CI/CD is applied:

- The project involves distributed teams across multiple institutes, requiring orchestration of work in a federated and collaborative way.
- The primary execution environment are HPC platforms, which is uncommon in traditional CI/CD practice [1].

- Beyond code management, the project anticipates extending CI/CD towards data processing and machine learning workflows, gradually incorporating concepts from *DataOps* and *MLOps* as the project matures.

This initial setup represents the first step in the overall project workflow, which will evolve to include advanced workflow automation, ML model life cycle management, and large-scale data processing.

2 DevOps Platform: GitLab

As part of providing a standard software development environment, SEANERGYS has decided to adopt GitLab as the central DevOps platform. GitLab is a DevOps platform that brings the full software lifecycle—planning, coding, testing, deployment, and monitoring—into a single application. At its core it provides Git-based source code management with familiar workflows such as branching, merge requests, and code reviews. Built-in CI/CD pipelines are defined directly in the repository, allowing builds, tests, and deployments to run without relying on external systems. The platform supports a hierarchical organisation of projects through groups and subgroups, making it easier to structure teams and manage permissions in larger or multi-disciplinary settings.

The execution layer is designed to be flexible. GitLab's runner architecture can operate on Docker, Kubernetes, or custom executors such as Jacamar, which makes it adaptable to a variety of infrastructures. The platform is also extensible, with Application Programming Interfaces (APIs) and integration points that enable the attachment of external applications or services. A further strength is its portability: organisations, groups, and projects can be moved between GitLab instances, whether Software-as-a-Service (SaaS) or self-managed, which provides freedom to adjust deployment strategies over time without locking into a single setup.

From a business perspective, GitLab follows an open-core model. A free edition supports broad adoption, while advanced functionality around scalability, governance, and enterprise support is provided through paid tiers. The platform can be consumed as SaaS or deployed in a self-managed configuration, giving organisations control over infrastructure and data when required. By combining source code management, automation, and collaboration tools in a unified system, GitLab reduces reliance on multiple disconnected platforms while maintaining flexibility and long-term adaptability.

Furthermore, the SEANERGYS project has selected the *Premium version of GitLab* for its advanced features that streamline our workflow, improve collaboration, and ensure traceability and reliability. In particular, we expect the following advantages from the Premium version:

Streamlined Development and Code Quality

- *Push rules* can automatically reject commits if they do not follow defined standards, such as static analysis results, commit message formats, or file types allowed in a repository.
- *Merge request guardrails* and *enforced approvers* ensure that changes are reviewed by designated maintainers before integration.
- *Code quality reports* highlight maintainability issues directly inside merge requests, supporting immediate feedback.

Efficient Project Management

- A *planning hierarchy* visualises epics and milestones on a timeline, helping identify dependencies across projects.
- *Epics* [2] group large initiatives, which can be promoted from individual issues to maintain context.

- *Issue weights* allow estimation of task complexity, while *iterations* support agile workflows by grouping tasks into time-boxed sprints.

Other Tools and Analytics

- *Remote repository pull mirroring* ensures robustness in the supply chain by automatically synchronising upstream changes.
- *Protected environments* restrict deployment access to authorised users, enforcing governance on sensitive stages.
- *Quality analytics* track test coverage, code quality, and pipeline performance to monitor long-term trends.

2.1 Hosting and Resources

SEANERGYS' GitLab instance will be hosted at JSC. We have chosen to self-host GitLab because this approach integrates more effectively with existing infrastructure and operational requirements than the SaaS option. Many of JSC's internal services are already connected to GitLab, which makes a self-managed instance easier to integrate directly without relying on external workarounds. A further consideration is *data residency*: with self-hosting, all project data, pipelines, and artifacts remain in the European Union (EU), ensuring compliance with local requirements and avoiding potential issues with data stored abroad.

Self-hosting also removes several restrictions that are present in cloud-hosted services:

Storage Cloud providers impose limits on storage for build artifacts and related data. With a self-hosted instance, storage can be scaled as needed without predefined quotas.

Runner execution SaaS platforms often enforce runtime limits for CI/CD runners. Hosting our own runners ensures that pipelines can run as long as required, without artificial cut-offs.

Connectivity Running GitLab inside our own network removes the need to expose custom runners or services through firewalls, making connections more reliable and secure.

In addition, self-hosting provides several technical advantages:

Instance-wide settings Centralised configuration of authentication methods, access policies, and project templates, with auditing capabilities.

CI/CD runners Dedicated runners that optimise performance and provide secure access to internal resources.

Performance and scalability Dedicated hardware that ensures consistent performance and allows resources to grow with project needs.

Access to institutional resources Direct integration with computing and storage infrastructure, as well as access to established backup and recovery processes.

Taken together, these factors make self-hosting the most practical and sustainable choice for our project. JSC already has experience operating a production GitLab instance that is connected to computing infrastructure and has been used successfully in projects such as JUPITER Research and Early Access Program (JUREAP). Hosting at JSC leverages this existing expertise, infrastructure, and access management, reducing the overhead and complexity of running our own GitLab instance.

2.1.1 Security and Governance

SEANERGYS will use established security and governance practices as guidelines for configuring and operating the project's GitLab instance. We will follow resources as listed below to provide principles and recommendations that will inform our setup:

- GitLab's official security documentation [3] and hardening guidance, outlining recommended configurations and operational safeguards.
- Community frameworks such as the Open Worldwide Application Security Project (OWASP) CI/CD Security Risks [4], which highlight common threats and mitigation strategies.
- Government and industry best practices such as the United States National Security Agency and the United States Cybersecurity and Infrastructure Security Agency report on defending CI/CD environments [5].

In addition, JSC has experience with production-grade authentication systems such as JuDoor, which we will leverage for secure access control on the SEANERGYS GitLab instance.

GitLab's extensible design also allows us to connect additional services while maintaining alignment with these guidelines. For example, the Jacamar CI framework for high-throughput workflows [6, 7] is one integration option.

2.2 Project Structure

Within GitLab, repositories are organised in a hierarchical scheme. In general, repositories should be placed in the top-level group. In particular shared code and common components are placed in this top-level group. Additionally, each Work Package (WP) will have its own group, which can be used as a playground. Initial experiments and very early designs should be started there. Later, they can be transferred into the top-level group. The partners can decide on a case-by-case basis when to migrate repositories. By default, all these repositories are visible to all partners. If required, exceptions can be made.

All participants are always free to create repositories, private or public, under their own user namespaces without having to ask for permissions. Even though participants are encouraged to move shared work into one of the shared WPs or the top-level group, there are no restrictions on private repositories.

To publish software and other documents, a special group `public` will be established. There, all repositories are visible to the world. To make a public release, developers can seamlessly push the current state into a repository inside the public group.

2.3 Current Status and Initial Deployment

Efforts to procure and deploy the Premium version of GitLab are currently underway and are expected to finish during September 2025. In the meantime, work has already started on deploying tools and infrastructure to support GitLab Premium when it is deployed. The current tools and development are using the existing GitLab instances at JSC, IT4I@VSB, and other partner institutes as a temporary measure till the SEANERGYS GitLab is up and running. These instances are already connected to the respective computing and storage infrastructures, allowing initial CI/CD experiments and development to commence. More details on the current snapshot of the development efforts are provided in Chapter 6.

3 CI/CD, DataOps, and MLOps Infrastructure

The CI/CD infrastructure for SEANERGYS will be built around GitLab, providing a unified framework for software development, testing, and deployment across all modules. This platform ensures reproducible environments, seamless integration with HPC and storage resources, and distributed workflows across partner institutes. Key features include automated versioning of datasets and models, traceable execution of pipelines, and integration with industry-standard model serving practices.

3.1 Pipeline and Testing Infrastructure

GitLab CI/CD is the core of our infrastructure, natively integrating with repositories, issue tracking, and code review tools. This allows us to define coherent, reproducible pipelines across all modules. Our setup will include:

Docker runners Lightweight, container-based runners will be provided for unit and functional tests. These are well-suited for rapid feedback during development, offering reproducibility and isolation while keeping turnaround times short. They enable developers to verify changes before scaling up to larger systems.

HPC runners Dedicated runners on JSC systems such as Jacamar integrate with the Slurm workload manager to execute large-scale or resource-intensive jobs. These runners allow the CI/CD pipeline to transparently submit, monitor, and collect results from HPC compute nodes, providing full traceability of job execution. This setup enables testing workflows under production-like conditions while still being managed within the GitLab environment. Through enforced identity propagation, the runners guarantee that CI/CD jobs inherit the full set of privileges and limitations associated with the submitting user's account on the target system.

SEANERGYS will provide reproducible environments through package management scripts, containerised setups, and pre-defined CI/CD components. This approach reduces redundant effort and ensures uniformity and coherence across testing environments. Results from CI/CD runs will be stored according to usage in either:

- Small outputs are retained as GitLab artifacts.
- Larger datasets and results use S3-compatible object storage integrated with GitLab.

Distributed execution will be enabled via GitLab-to-GitLab triggers, allowing pipelines to run at partner institutes with centralised pipeline tracking and collection at SEANERGYS gitlab. Distributed execution refers to running CI/CD jobs across multiple GitLab instances. Using GitLab-to-GitLab triggers, a pipeline step in one repository can start a job on a partner's instance, executed on their local infrastructure under their own policies. The central pipeline then collects job status and artifacts, enabling unified tracking. A key feature is that partner sites do not need to establish direct connections to the SEANERGYS GitLab or modify their existing processes. This approach is related to GitLab's multi-project (or downstream) pipelines [8, 9] but extends across different GitLab instances at separate sites.

As an example, a commit in the central repository may trigger:

- a partner site running data monitoring and producing a dataset,
- another training an ML model on local cluster,
- a third running optimisation on the model outputs on yet another partner site.

The central pipeline aggregates results and artifacts, while execution remains local at each site.

3.2 Deployment, Integration, and Model Serving

The CI/CD infrastructure in SEANERGYS has been deliberately designed to go beyond testing, supporting deployment and integration of both software components and machine learning models. SEANERGYS chooses to couple pipelines with storage and service platforms, so that models and services can be deployed in a reproducible and traceable manner across diverse environments.

A central integration repository was selected as the single point of coordination, providing a consistent view of the software stack and hosting the integration test infrastructure. This design ensures:

Unified deployment SEANERGYS opts for full-stack deployment across modules and institutes, rather than isolated pipelines.

Systematic validation Integration reviews are required to verify compatibility and correctness at each stage.

Automated release flow Releases for both software and ML models are managed directly via CI/CD to avoid manual overhead.

Built-in documentation We decided to generate project documentation via CI, ensuring references remain consistent and up to date.

In terms of benefits, these choices provide:

Incremental evolution The platform was designed to adapt to module-specific requirements without disrupting existing pipelines.

Extensible testing and deployment: SEANERGYS intentionally allows new validation routines to be plugged into the CI/CD structure.

Centralised utilities Common scripts, templates, and configurations are consolidated to reduce duplication and ease reuse.

HPC and cloud integration SEANERGYS chooses to integrate directly with both HPC and cloud platforms, ensuring scalability from routine jobs to large-scale production runs.

Overall, these design decisions establish a deployment and serving strategy that is reliable, reproducible, and transparent, while preserving the flexibility to expand and adapt with project needs.

3.3 DataOps

DataOps extends the principles of DevOps to data engineering. While DevOps focuses on CI/CD for software artifacts (typically small binaries or services), DataOps applies the same automation mindset to the construction and operation of *data pipelines*.

In software CI/CD, the input is source code and the pipeline executes tasks such as compiling, testing, and deployment. In contrast, DataOps pipelines usually begin with data ingestion, followed by stages such as extraction, transformation, validation, quality checks, and ultimately storage or publishing. The outputs are often large datasets, tables, or feature stores rather than compact binaries.

The core idea is that when pipeline definitions or transformation code changes, the entire data flow is automatically re-executed. This includes reprocessing, validating, and publishing the data, thus avoiding manual intervention such as one-off cleaning or loading.

Compared to DevOps, DataOps introduces additional challenges in connection with SEANERGYS:

Data scale and versioning Datasets can be orders of magnitude larger than software artifacts. Managing their versions requires specialised approaches, often connecting Git with external object storage. Examples include Git Large File Storage (LFS), Data Version Control (DVC), git-annex, or datalad. Ensuring the operations of such infrastructure itself has to be a part of CI/CD operations for a project like SEANERGYS.

Infrastructure testing Pipelines depend on external infrastructure like databases, caches, distributed storage. Proper operations of workflows, for example with regards to caching, is important to ensure that the CI/CD pipelines themselves can run correctly and efficiently. Continuous testing must therefore include code correctness, performance and infrastructure availability of the data in addition to software testing.

Data quality and evolution Beyond code regressions, DataOps must guard against data drift, schema changes, and inconsistencies in upstream sources.

Reproducibility Since data sources may evolve over time, DataOps emphasises the ability to reproduce past states of both pipelines and datasets.

DataOps and MLOps are closely connected. High-quality, reproducible, and validated datasets produced through DataOps pipelines form the foundation for reliable MLOps practices. In practice, DataOps ensures that model training and deployment pipelines always receive consistent and trustworthy input data. The interplay between MLOps and DataOps, and their interaction with CI/CD will be further defined in the coming months in collaboration with the other WPs.

Similarly, testing and validation of code managed by the DevOps platform requires reliable access to data (such as recorded monitoring data and required output); DataOps will ensure that such data is made available, with the precise interaction with the Data Plane components to be further defined.

3.4 Machine Learning Operations (MLOps)

Machine learning is a core component of SEANERGYS. MLOps extend the CI/CD paradigm by providing workflows, infrastructure, and practices that ensure ML model development, deployment, and monitoring are **reproducible, automated, and traceable**.

In contrast to traditional software CI/CD pipelines—which mainly take code as input and execute steps such as compiling, testing, and deploying—ML pipelines must also handle **large datasets and trained models**. A typical ML CI/CD workflow therefore includes steps such as:

- Hyperparameter tuning
- Model training
- Model testing and evaluation
- Deployment to an inference server

To support these workflows, established tools such as MLflow [10] can be leveraged to streamline experiment tracking, hyperparameter sweeps, training monitoring, and inference deployment.

Within SEANERGYS, a *Model Zoo* acts as a *continuous output* of these MLOps pipelines. It provides a shared repository of pre-trained and project-specific models, developed and published according to prevailing industry standards (e.g., TensorFlow Hub [11], PyTorch Hub [12]). The Model Zoo stores:

- Model architectures and pre-trained weights,
- Metadata: input/output specifications, accuracy scores, training datasets or references,
- Inference APIs, utility scripts, and documentation for reuse and reproducibility.

Roles in the model life cycle include:

- **Experimentation**: Reuse of pre-trained models to reduce training costs
- **Training**: Fine-tuning models on new datasets
- **Versioning**: Tracking models and associated data
- **Serving**: Deploying via APIs or registries
- **Collaboration**: Enabling cross-work-package sharing of models

Training pipelines are orchestrated through GitLab CI, with computationally intensive workloads executed on HPC systems. Models, logs, and metrics are published to the Model Zoo to ensure reproducibility, while metadata (e.g., job identifiers, node usage) are captured for traceability. Deployment pipelines leverage scalable backends to provide inference capabilities, with quality gates enforcing validation criteria before promoting models from staging to production.

The concept of MLOps further enables **evaluating pipelines by their degree of automation**. Mature pipelines incorporate:

- Performance evaluation at multiple stages

- Conditional branching based on metric thresholds
- Management of diverse objects (model repositories, compute environments, datasets)
- Automated metric collection and decision-making

This allows teams to continuously integrate and deliver ML models with the same rigor as software, while accounting for the added complexity of data, models, and experimentation. For further reference, see the MLOps maturity model of Microsoft Azure [13] and Google's ML pipeline automation framework [14].

Training pipelines will be orchestrated through GitLab CI, with heavy computation executed on HPC systems. Models, logs, and metrics will be published to the Model Zoo to ensure reproducibility. Metadata such as job identifiers and node usage will be captured to maintain traceability. Deployment pipelines leverage scalable backends to provide inference capabilities, while quality gates enforce validation criteria before promoting models from staging to production.

Observability for MLOps

Observability is essential to ensure that ML models remain reproducible, reliable, and performant throughout their lifecycle. In the context of MLOps, observability spans several complementary layers:

Reproducibility factors Capturing hyperparameters, dataset versions, code revisions, and random seeds to ensure that training runs can be faithfully repeated.

Learning dynamics Recording training and validation curves across iterations (e.g., loss, accuracy evolution) to provide insight into convergence and potential overfitting.

Generalization capability Monitoring metrics such as accuracy, F1-score, mean squared error (MSE), and other task-specific indicators to assess real-world performance.

System metrics Tracking infrastructure-level indicators such as Central Processing Unit (CPU)/Graphics Processing Unit (GPU) utilisation, memory consumption, Input / Output (I/O) throughput, and queue wait times to evaluate efficiency and scalability.

Inference metrics Measuring production performance including model drift, inference latency, request throughput, and deployment frequency to ensure models remain effective after release.

These metrics will be visualised in dashboards that integrate training performance, system utilisation, and inference health. This enables developers and operators to monitor not only the technical correctness of models but also their operational behaviour in real time.

SEANERGYS emphasises transparency, reproducibility, and portability. Traceability will be maintained via model lineage, audit logs, and version history. Containerisation ensures isolation and portability across HPC and cloud platforms:

- Docker and Podman for general development.
- Singularity/Apptainer for HPC environments.

This approach ensures that models and workflows remain consistent, operating system-agnostic, and deployable across infrastructures.

4 Planning and Tracking Work

GitLab offers a variety of features to plan, organise and track collaborative work. In the following, the most important ones are outlined and put in the context of SEANERGYS's work. GitLab provides extensive documentation on planning and tracking work in their online documentation [15]. On top of that, this chapter gives an outline on Architecture Decision Records (ADRs) as a complementary planning concept and explains how they are going to be used in SEANERGYS with GitLab.

4.1 GitLab Work Items

4.1.1 Issues and tasks

GitLab issues [16] are the core work item unit of GitLab to plan work, assign work to people, track progress, propose new features, and report bugs. In SEANERGYS they will serve as the basic means for collaborative work on software components, work packages and across the work packages. They contain descriptions of required work, proposals for new features, bug reports, or documentation about findings obtained in ongoing work. GitLab issues will be linked with any related other GitLab work items such as other issue(s), merge request(s), etc. to ensure that cross-links are tracked properly. One or multiple persons can be assigned to an issue to indicate responsibilities. In addition, any person interested in an issue can enable email notifications about changes to stay up-to-date and get involved on demand.

In case a GitLab issue has multiple smaller subtasks, GitLab provides tasks [17] as sub-units to issues to further refine the work planning and assignment. SEANERGYS will use GitLab tasks as needed for a fine-granular division, discussion, and assignment of work within an issue.

4.1.2 Epics

GitLab's epics [2] are a feature only available in the Premium and Ultimate versions. They provide means to coordinate multiple GitLab work items into a hierarchy and enable the coordination and organisation of large and complex work plans. As shown in Fig. 1, GitLab epics can be organised in hierarchies themselves (parent and child epics). GitLab issues and GitLab tasks are "contained" in a GitLab epic. Furthermore, relationships between GitLab issues/tasks and objectives and results of an epic can be represented. Three types of relationships [18] can be represented: "relates to", "blocks", and "is blocked by". These three types of explicit relationships can also implicitly define relationships between different GitLab items in a hierarchical GitLab epic structure.

SEANERGYS will use GitLab epics whenever the need for coordination of development work across work packages arises. SEANERGYS' epics will provide a global overview for the project consortium on the status of the ongoing work.

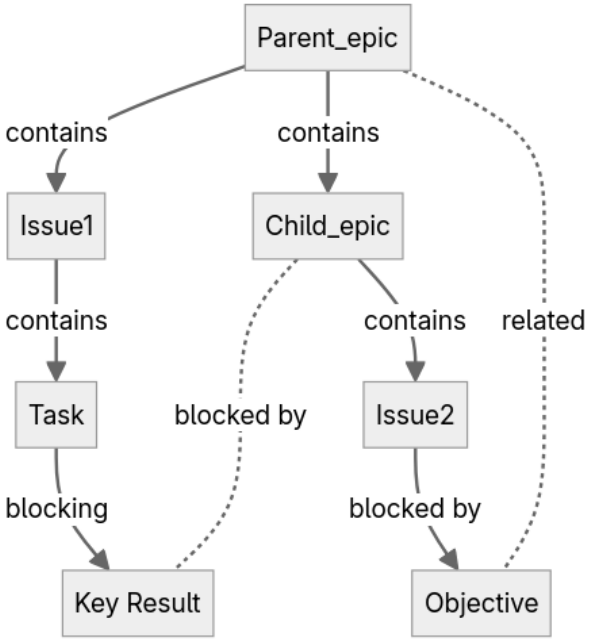

Figure 1: Example set of relationships of GitLab epics [2]. Solid line: Explicit relationship; Dotted line: Implicit relationship.

4.2 Labels

GitLab labels [19] can help to categorise and prioritise GitLab issues, merge requests and epics. One or multiple GitLab labels can be assigned to each GitLab work item to assign properties to them and visualise these properties by colours. They help filtering GitLab work items for certain topics and properties such as priorities. GitLab labels can be defined per GitLab project or group so that there can be labels specific per component, work package and the top-level SEANERGYS GitLab group. With GitLab Premium and Ultimate licences, GitLab labels can be scoped to offer mutually exclusive labels, e.g., a label with the scope `state` could be one of `state:ongoing`, or `state:reviewing`, or `state:accepted`.

4.3 Milestones

GitLab milestones [20] can be defined in GitLab per project or group to provide another means of work organisation. They support progress tracking towards a certain goal. Multiple GitLab work items, merge requests, participants, and labels belong to a GitLab milestone and define the overall goal. In the SEANERGYS context, such a goal can be a new release of a specific software component, the preparation of a demonstrator for (parts of) a WP's developments, or a project deliverable or milestone that needs to be achieved at a certain time. GitLab milestones can support time planning of the collaborative work by including start and due dates.

4.4 Architecture Decision Records

ADRs[1] are lightweight documents that capture significant technical and architectural decisions, the context in which they were made, and their consequences. Teams typically write an ADR for decisions that are costly to reverse or broadly impactful, e.g., system decomposition, data storage choices, messaging patterns, security posture, or standards that affect many components or teams. ADRs are deliberately kept brief, such that they can serve as a reference for future discussions or new team members.

Although there is no fixed form for ADRs, developers are encouraged to choose a common template for their project. As an example, Fig. 2 shows the full structure of the Markdown Architectural Decision Record (MADR) [21] template. The author notes that in cases where less information is sufficient, the template can be reduced to the items *Title*, *Context and Problem Statement*, *Considered Options*, *Decision Outcome (with Justification)*, and *Consequences*; bringing it close to the popular template from Nygard [22].

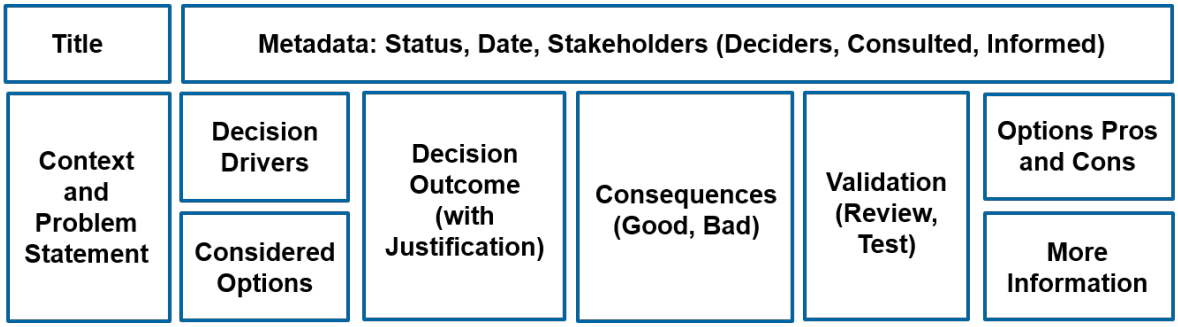

Figure 2: Full MADR template structure, visualisation from [21]

ADRs are stored as markdown files alongside the code base. As such, they can utilise the same tools as source code for drafting and discussions, e.g., GitLab issues and merge requests, linking them to established requirements. Once accepted, ADRs become immutable, allowing them to be referenced in future discussions in GitLab issues or merge requests. Unlike transient meeting notes or wikis, ADRs are reviewed like code, and are superseded by newer ADRs rather than rewritten, preserving the rationale over time. These links of code changes and/or GitLab issues to ADRs, combined with the links from ADRs to requirements, enable full traceability from requirement to source code.

In the development life cycle, ADRs are part of the architecture design process. This makes ADRs inherently scoped: Each (sub-) system designed within SEANERGYS can maintain its own set of ADRs (also: Architecture Decision Log (ADL)). Partners with power over the respective system can equally contribute to this system's ADRs. We will at least establish and maintain ADRs for the global architecture as part of WP1. Beyond that, each WP can use GitLab to collect their own ADRs in their repositories.

We will prepare training material on the establishment and continued maintenance of ADRs in GitLab as part of the ongoing work in WP5. Close collaboration with the other WPs, in particular WP1, will ensure that the training material matches the demands in SEANERGYS. On a technical level, SEANERGYS will

[1]The terms *Architecture* and *Architectural* in ADR are used interchangeably by different sources. This document uses the term *Architecture Decision Record (ADR)*.

employ established standards for ADRs such the templates proposed by Zimmermann [21] or Nygard [22], and tooling such as `log4brains` [23] or `adr-tools` [24].

5 Metrics and Observability

Observability is a fundamental component of the CI/CD, DataOps, and MLOps infrastructure. It enables monitoring of software delivery, infrastructure usage, and machine learning workflows, ensuring reproducibility, auditability, and performance optimisation. In this deliverable, the emphasis is on the types of metrics to collect, access mechanisms, storage infrastructure, and linkage to jobs and models, rather than the exact metric definitions. The actual definitions of (code) quality criteria and their metrics will be provided in SEANERGYS' Deliverable D5.2.

5.1 Types of Metrics

The platform captures metrics across multiple dimensions, reflecting both development and operational performance. These include:

Code quality metrics Derived through static and dynamic code analysis to assess code quality and aid code reviews. Examples include test and documentation coverage, type checks, or cyclomatic complexity.

Software delivery metrics Collected by CI/CD platforms and DevOps tools to monitor development throughput, deployment reliability, and release efficiency. Examples include deployment frequency, pipeline success/failure rates, and time from commit to deployment.

Data quality metrics Computed as part of the DataOps workflow to assess the quality of processed datasets. Examples include Volume, Variety, and Variability. As these data sets may also be ingested by ML models during training, these metrics do not only concern DataOps, but are also used for MLOps.

Infrastructure metrics Monitored by cloud platforms, HPC clusters, and orchestration systems to ensure efficient resource usage and detect performance bottlenecks. Examples include CPU/GPU utilisation, memory usage, and queue wait times.

ML workflow metrics Tracked by ML platforms and pipeline orchestration tools to evaluate training progress, model performance, and production stability. Examples include model training progress, inference latency, and model drift indicators.

By combining these perspectives, teams can assess the efficiency of the development process, quality of data sets, and the reliability of deployed ML models.

5.2 Access and Collection

Metrics are gathered through programmatic interfaces and runtime logs, allowing association with individual jobs, pipelines, or model versions. Key access methods include:

- GitLab REST and GraphQL APIs for repository and pipeline activity.

- CI job logs and container outputs for runtime information.
- Monitoring tools and frameworks to extract system and ML-specific metrics.

This approach ensures traceability, reproducibility, and flexibility in how metrics are captured and used. For example, key code quality metrics can be associated to a proposed code change, aiding reviewers.

5.3 Storage and Association

To make metrics actionable, they must be stored reliably and associated with their originating workflows. The infrastructure will support:

- Centralised, scalable databases or time-series stores for structured storage.
- Durable, replicated storage for long-term retention and reproducibility.
- Linking of metrics to specific jobs, pipelines, or model versions to enable analysis, aggregation, and visualisation.

Stored metrics can feed dashboards and reporting tools, providing both real-time insights and historical context for auditing and optimisation.

6 CI/CD Demonstrator

A minimal demonstrator has been created to validate one of the ideas that are proposed for the the SEANERGYS development platform and to show that the ideas can be operational and support essential CI/CD workflows. The demonstrator illustrates core capabilities such as containerised execution, cross-GitLab triggering, and storage of job results.

6.1 Repository Overview

This section provides a brief overview of the repositories created for the demonstrator. All repositories are tagged with version `d5.1`, establishing a fixed reference version for use in the demonstrator.

6.1.1 Python example repository `pyexample`

URL `https://gitlab.jsc.fz-juelich.de/seanergys/wp5-ci-cd-and-quality-assurance/software/pyexample`

Git Tag `d5.1`

This is a Python example repository that follows best practices by using `pyproject.toml` for project configuration, dependency management, and build specification. It includes a simple script for demonstration purposes, unit tests to validate correctness, and a ‘pyproject.toml‘. This setup provides a lightweight example for CI/CD validation while remaining simple to understand and deploy. Furthermore, the project is configured to also run code formatting and linter/quality checks, and generate its own docs.

6.1.2 Python CI/CD component

URL `https://gitlab.jsc.fz-juelich.de/seanergys/wp5-ci-cd-and-quality-assurance/software/catalog/python`

Tag `d5.1`

This repository provides reusable CI/CD components for Python projects in SEANERGYS. It defines standardised pipelines for building, testing, type-checking, code formatting, and license compliance checks. Each pipeline runs in a controlled Python environment using hatch and supports multiple Python versions via semantic versioning, with legacy, current, and next toolchain versions. These components ensure consistent, reproducible, and maintainable CI/CD workflows across projects and preventing redundant work.

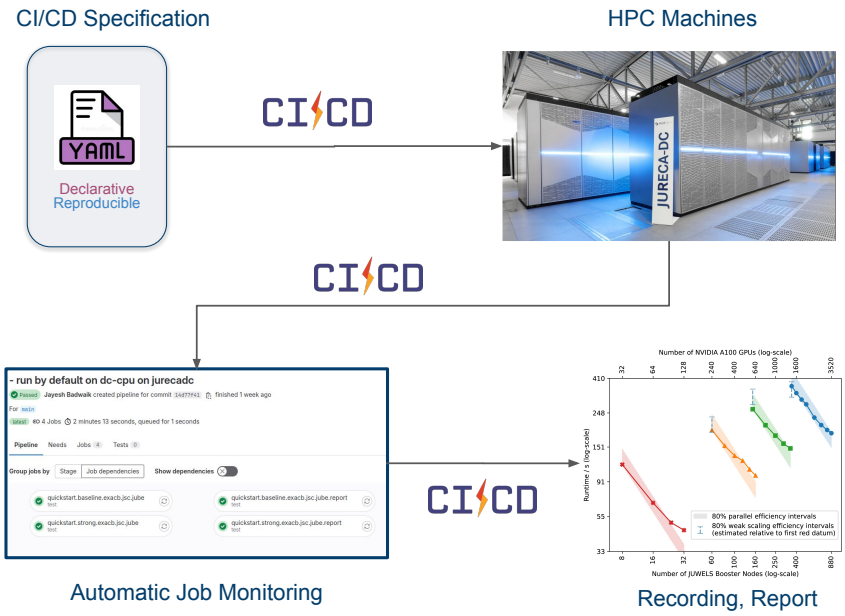

FIGURE 3: OVERVIEW OF THE PYTHON CI/CD DEMONSTRATOR IN SEANERGYS

6.2 Demonstrator

The Python demonstrator illustrates a fully declarative CI/CD workflow within the SEANERGYS platform. Users define a `.gitlab-ci.yml` file that specifies the pipeline entirely in a declarative manner, without requiring imperative scripting. This configuration enables automated job execution both within Docker containers and on JSC HPC machines, with minimal adjustments to parameters when switching environments.

The CI jobs perform tasks such as unit testing, type checking, code formatting, and license verification. After execution, the results are collected and uploaded to the GitLab repository in an orphan branch as JavaScript Object Notation (JSON) files. This ensures reproducible, traceable, and environment-independent outcomes while keeping the main branch clean.

Fig. 3 shows the overall workflow, from declarative pipeline definition to execution and result storage across diverse compute environments.

Fig. 4 shows an example job execution using this template. The jobs with `docker` in their name use the docker executor, `jsc` indicates execution on JSC HPC machines. The job `python.jsc.test` `python.jsc.test` failed due to missing the required minimum test coverage. As later jobs require this job to pass, they are marked as skipped in Fig. 4.

6.2.1 Running the CI job on JURECA machine

In the following snippet from `.gitlab-ci.yml`, the `jsc` component is used to select the target machine for running the CI job. A unique prefix `jsc.shell` is prepended automatically by the component to all jobs it generates.

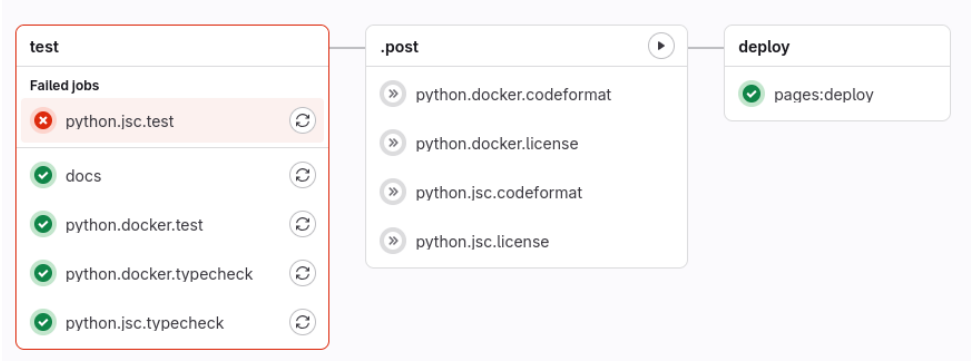

Figure 4: Example CI pipeline of the Python CI/CD demonstrator in SEANERGYS, here demonstrated on the example repository from Section 6.1.2

The configuration for running the CI job is defined in the template job `.<prefix>.select`, where in this case `<prefix>` is `.jsc.shell`. This template is passed as a fixture to our `commit` component, which is responsible for executing tests that should run on every commit.

By creating a job that extends the `.jsc.shell.select` template, the job is automatically scheduled on the Jülich Research on Exascale Cluster Architectures (JURECA) machine with the correct compute project and budget configuration, ensuring reproducible and resource-aware execution.

```yaml
include:
  - component:
    ↪ gitlab.jsc.fz-juelich.de/seanergys/wp5-ci-cd-and-quality-assurance/
    ↪ software/catalog/sites/jsc@main
    inputs:
      prefix: "jsc.shell"
      machine: "jureca"
      runner: "shell"
      project: "cjsc"
      budget: "zam"
  - component:
    ↪ gitlab.jsc.fz-juelich.de/seanergys/wp5-ci-cd-and-quality-assurance/
    ↪ software/catalog/python/commit@main
    inputs:
      prefix: "python.jsc"
      version: "latest"
      fixture: .jsc.shell.select
```

This is included in the example shown in Fig. 4 above. There, the following jobs are configured to run on JURECA; their state in the example Fig. 4 is denoted in parentheses:

- `python.jsc.typecheck` (succeeded)
- `python.jsc.test` (failed)
- `python.jsc.license` (skipped)
- `python.jsc.codeformat` (skipped)

6.2.2 Running the CI job on Docker

In the following snippet from `.gitlab-ci.yml`, the `docker` component is used to select the target machine for running the CI job. We provide a unique prefix, `docker`, which the component automatically prepends to all jobs it generates.

```
include:
  - component:
    ↪ gitlab.jsc.fz-juelich.de/seanergys/wp5-ci-cd-and-quality-assurance/
    ↪ software/catalog/sites/jsc@main
    inputs:
      prefix: "docker"
  - component:
    ↪ gitlab.jsc.fz-juelich.de/seanergys/wp5-ci-cd-and-quality-assurance/
    ↪ software/catalog/python/commit@main
    inputs:
      prefix: "python.docker"
      version: "latest"
      fixture: .docker.select
```

The Docker setup is nearly identical to the JURECA configuration. The only changes required are the site component and the job prefix, demonstrating the modularity and reusability of our CI/CD components across different compute environments.

7 Conclusion

7.1 Current Status

The development platform of SEANERGYS is GitLab, and it is hosted at JSC. This leverages the user access and storage management of JSC so that accounts can be managed via the well-established access portal JuDoor.

An initial hierarchical group and project structure, and large parts of the DevOps capabilities of GitLab are already accessible to the SEANERGYS project consortium. GitLab issues, tasks, labels, and milestones can be used to plan, organise, assign, and track the project's development work. ADRs are lightweight markdown documents used to capture significant technical and architectural decisions within the SEANERGYS project. These ADRs are stored in GitLab alongside the code base, ensuring traceability from requirements to source code and supporting transparency for all stakeholders.

Significant preparations have been made to use GitLab's CI/CD capabilities to support and enhance the development efforts of SEANERGYS. For this purpose, a demonstrator for basic CI/CD stages has been set up. This demonstrator will serve as a basis to integrate CI/CD into the upcoming development efforts.

Since ML is a core component of SEANERGYS, GitLab's capabilities to support MLOps have been explored extensively and put into the context of the project's ML Model Zoo as a foundation for future steps. For monitoring of the development efforts via CI/CD and the MLOps infrastructure of SEANERGYS query interfaces and a first potential set of metrics have been identified successfully.

7.2 Next Steps

The current GitLab instance uses a non-premium license that limits the features and prevents full usage of all capabilities in SEANERGYS. The procurement of a Premium GitLab license for the project is in progress at the time of writing this document. Once the Premium license is available, the current status and solutions including the project structure and repositories need to be transferred to the Premium SEANERGYS GitLab instance. In the course of this activity, access policies (two-factor authentication, single sign-on) need to be finalised and communicated with partners. Furthermore, GitLab epic work items need to be created that map to cross-work package development efforts of SEANERGYS.

For the CI/CD, an approach for artifact storage and fitting backup procedures needs to be finalised and implemented. On top of that, HPC runners need to be integrated into the GitLab Premium setup. Based on the demonstrator, the CI jobs and deployment stages for SEANERGYS will be extended along with ongoing development work. The demonstrator will be connected to an integration repository to support this process.

In the weeks after the submission of this document, the project's hierarchical GitLab structure will be finalised. It will be clarified when and how repositories are promoted from work package groups to the top-level group. This will help ensure consistency as components mature and are integrated into the SEANERGYS software stack.

The project will develop training material on the establishment and maintenance of ADRs to ensure all team members are familiar with the process. Collaboration with other work packages, particularly WP1, will be prioritised to align ADR practices and training with overall project needs.

Regarding monitoring of the development efforts and progress, the required metrics for internal monitoring and reporting to EuroHPC need to be identified (Deliverable D5.2). Furthermore, storage location, access control, collection frequency, level of collection automation, and means of visualisation for monitoring data need to be decided.

Acronyms and Abbreviations

A

ADL *(Architecture Decision Log)* 19
ADR *(Architecture Decision Record)* 17, 19, 20, 27, 28
API *(Application Programming Interface)* 8, 15, 21

C

CI/CD *(Continuous Integration and Continous Deployment)* 5–15, 21, 23, 24, 26, 27
CPU *(Central Processing Unit)* 16, 21

D

DVC *(Data Version Control)* 14

E

EU *(European Union)* 9

G

GPU *(Graphics Processing Unit)* 16, 21

H

HPC *(High Performance Computing)* 5, 6, 12, 13, 15, 16, 21, 24, 27

I

I/O *(Input/Output)* 16
IT4I *(IT4Innovations)* IT4Innovations national supercomputing center 5, 11

J

JSC *(Jülich Supercomputing Centre)* Jülich Supercomputing Centre GmbH, Jülich, Germany 5, 9–12, 24, 27, 29
JSON *(JavaScript Object Notation)* 24
JuDoor Portal for managing accounts, projects and resources at JSC 10, 27
JUREAP *(JUPITER Research and Early Access Program)* 10
JURECA *(Jülich Research on Exascale Cluster Architectures)* 25, 26

L

LFS *(Large File Storage)* 14

M

MADR *(Markdown Architectural Decision Record)* 19
ML *(Machine Learning)* 6, 7, 13, 15, 16, 21, 27
MLOps *(Machine Learning Operations)* 6, 14–16, 21, 27
MSE *(mean squared error)* 16

O

OWASP *(Open Worldwide Application Security Project)* 10

R

Bibliography

[1] T. Gamblin and D. S. Katz. "Overcoming Challenges to Continuous Integration in HPC". In: *Computing in Science & Engineering* 24.6 (Nov. 2022), pp. 54–59. ISSN: 1558-366X. DOI: 10.1109/mcse.2023.3263458. URL: http://dx.doi.org/10.1109/MCSE.2023.3263458.

[2] *GitLab Epics Documentation*. URL: https://docs.gitlab.com/user/group/epics/.

[3] *GitLab Security Documentation*. URL: https://docs.gitlab.com/security/.

[4] OWASP. *OWASP Top 10 CI/CD Security Risks*. URL: https://owasp.org/www-project-top-10-ci-cd-security-risks/.

[5] U. S. National Security Agency and U. S. Cybersecurity and Infrastructure Security Agency. *Defending Continuous Integration/Continuous Delivery (CI/CD) Environments*. 2023. URL: https://media.defense.gov/2023/Jun/28/2003249466/-1/-1/0/CSI_DEFENDING_CI_CD_ENVIRONMENTS.PDF.

[6] P. Bryant. *Jacamar CI Documentation*. URL: https://ecp-ci.gitlab.io/index.html.

[7] P. Bryant. *Jacamar CI*. URL: https://gitlab.com/ecp-ci/jacamar-ci.

[8] *GitLab Downstream Pipelines Documentation*. URL: https://docs.gitlab.com/ci/pipelines/downstream_pipelines/.

[9] F. Pitino. *Breaking down CI/CD complexity with parent-child and multi-project pipelines*. Feb. 22, 2022. URL: https://about.gitlab.com/blog/parent-child-vs-multi-project-pipelines/.

[10] *MLflow*. URL: https://mlflow.org/.

[11] *TensorFlow Hub*. URL: https://www.tensorflow.org/hub.

[12] *PyTorch Hub*. URL: https://pytorch.org/hub/.

[13] Microsoft Azure Documentation. *Machine Learning operations maturity model*. URL: https://learn.microsoft.com/en-us/azure/architecture/ai-ml/guide/mlops-maturity-model.

[14] Google Cloud Documentation. *MLOps: Continuous delivery and automation pipelines in machine learning*. Aug. 28, 2024. URL: https://cloud.google.com/architecture/mlops-continuous-delivery-and-automation-pipelines-in-machine-learning.

[15] *GitLab Docs*. URL: https://docs.gitlab.com/.

[16] *GitLab Issues Documentation*. URL: https://docs.gitlab.com/user/project/issues/.

[17] *GitLab Tasks Documentation*. URL: https://docs.gitlab.com/user/tasks/.

[18] *GitLab Linked Epics Documentation*. URL: https://docs.gitlab.com/user/group/epics/linked_epics/.

[19] *GitLab Labels Documentation*. URL: https://docs.gitlab.com/user/project/labels/.

[20] *GitLab Milestones Documentation*. URL: https://docs.gitlab.com/user/project/milestones/.

[21] O. Zimmermann. *The Markdown ADR (MADR) Template Explained and Distilled*. Aug. 30, 2024. URL: https://www.ozimmer.ch/practices/2022/11/22/MADRTemplatePrimer.html.

[22] M. Nygard. *Documenting Architecture Decisions*. Nov. 15, 2011. URL: https://cognitect.com/blog/2011/11/15/documenting-architecture-decisions.

[23] T. Vaillant et al. *Log4brains*. Dec. 17, 2024. URL: https://github.com/thomvaill/log4brains.

[24] N. Pryce et al. *ADR Tools*. Mar. 30, 2020. URL: https://github.com/npryce/adr-tools.