

**HORIZON EUROPE PROGRAMME - HORIZON-CL5-2023-D3-01-01**

*Renewable Energy Valleys to increase energy security while accelerating the green transition in Europe - Innovation action (IA)*



**REFORMERS**  
RENEWABLE ENERGY VALLEYS

**REFORMERS**

Regional Ecosystems **FOR** Multiple-Energy Resilient Systems

Grant Agreement No. 101136211

Duration: 60 months | 1st November 2023 - 31st October 2028

**D5.3: MODEL LIFECYCLE FOR DIGITAL TWINS**

---



Funded by  
the European Union

## DOCUMENT INFO

---

**Deliverable number** D5.3

**Deliverable title** Model lifecycle for digital twins

**Work Package** WP5

**Deliverable type** Report

**Dissemination level** Public

**Due date** M24 (October 2025), Extended to M26 (December 2025)

**Pages** 32

**Document version** 3.0

**Lead author(s)** Edmund Widl (AIT)

**Contributors** Marc Dünser (AIT), Jort Groen (TUD)



**Funded by  
the European Union**

**Project funded by**



Schweizerische Eidgenossenschaft  
Confédération suisse  
Confederazione Svizzera  
Confederaziun svizra

Swiss Confederation

Federal Department of Economic Affairs,  
Education and Research EAER  
**State Secretariat for Education,  
Research and Innovation SERI**

## DOCUMENT CHANGE HISTORY

Version	Date	Author	Description
<b>DRAFT</b>			
0.1	21.07.2025	E. Widl, AIT	Structure, initial contents
0.2	18.11.2025	E. Widl, AIT	Version for internal review
<b>FIRST REVIEW</b>			
1.0	28.11.2025	P. Tzallas, CERTH	Proof reading and peer review
1.1	09.12.2025	R. Jastrzebski, WTG	Proof reading and peer review
1.2	12.12.2025	E. Widl, AIT	Consolidation of input from reviewers
<b>COORDINATOR APPROVAL</b>			
2.0	16.12.2025	T. Coosemans (VUB)	Coordinator review
<b>FINAL VERSION</b>			
3.0	16.12.2025	S. Arapoglou (VUB)	Format review, version ready for submission



## EXECUTIVE SUMMARY

This document presents the methodology and technical framework for generating, deploying, and managing digital twin models within the REFORMERS project. Building upon the high-level design requirements introduced in Deliverable D5.1 (*Digital Twin Design Requirements and Initial Architecture*), it defines a set of technical requirements and a lifecycle approach for models that support the REFORMERS Digital Twin (DT) ecosystem. The goal is to enable the automated, consistent, and scalable creation of diverse models with varying resolution, complexity, and energy vectors (heat, electricity, gas) to power the services offered by the REFORMERS DT.

At the core of the methodology is the **digital twin lifecycle**, comprising planning, design, validation, operation, and evolution phases. These stages guide model development from initial conception to iterative refinement and, ultimately, retirement. This lifecycle acknowledges the heterogeneity of models developed by various consortium partners and therefore avoids imposing a single modelling approach. Instead, a common set of **technical requirements** ensures interoperability and quality across all models. These requirements emphasize modularity, reusability, security-by-design, scalable data handling, flexible algorithm integration, and support for diverse communication protocols.

To implement these requirements effectively, this document introduces a three-component **technical framework**:

- **Common Runtime Environment:** The Rapid Deployment Platform (RDP) acts as the common environment for the execution of all REFORMERS DT models and services. It provides containerized, modular components for data retrieval, real-time data exchange, long-term storage, messaging, security, and field-device communication. This environment supports reliable prototyping and deployment across different Renewable Energy Valleys (REV) and operational contexts.
- **Automated Model Generation Environment:** This environment bridges model development and operational deployment by automating the creation of executable models and enforcing consistent metadata labelling. It introduces **model generators** and **metagenerators** that package model source code, containerization instructions, and provenance metadata into reproducible, traceable artifacts. A **Model API & Container Registry** hosts all generators and models and provides an interface for discovering, validating, and creating models on demand. The result is a robust basis for DevOps-driven continuous deployment and lifecycle tracking.
- **Knowledge Graph Integration:** The knowledge graph, described in Deliverable D5.2, provides standardized, linked metadata for assets and processes in REVs. It serves as a unified information source that models and services can query to configure themselves automatically, reducing manual configuration and ensuring semantic consistency across services.

Finally, the deliverable demonstrates the full workflow through a detailed **proof-of-concept implementation**. This example of a wind power forecasting service integrates asset data



from the knowledge graph, automates model creation from an existing forecasting tool (EOLICA), and deploys the resulting model using DevOps pipelines. It illustrates how the technical framework enables seamless, reproducible generation and deployment of REFORMERS DT models and services in realistic operational settings.

Overall, this document establishes a comprehensive lifecycle and technical foundation for scalable, automated, and interoperable digital twin modelling within the REFORMERS project, ensuring that diverse models can evolve consistently while supporting real-world energy innovation across multiple sites.



**Funded by  
the European Union**

**Project funded by**



Schweizerische Eidgenossenschaft  
Confédération suisse  
Confederazione Svizzera  
Confederaziun svizra

Swiss Confederation

Federal Department of Economic Affairs,  
Education and Research EAER  
**State Secretariat for Education,  
Research and Innovation SERI**

## TABLE OF CONTENTS

---

<b>LIST OF FIGURES .....</b>	<b>6</b>
<b>1. INTRODUCTION.....</b>	<b>8</b>
<b>2. DIGITAL TWIN TECHNICAL REQUIREMENTS .....</b>	<b>11</b>
<b>3. RUNTIME ENVIRONMENT FOR MODELS AND SERVICES.....</b>	<b>13</b>
3.1 General RDP Architecture .....	13
3.2 REFORMERS DT Services and Models Architecture .....	14
3.3 Decentralized REFORMERS DT Setups .....	15
<b>4. AUTOMATED MODEL GENERATION ENVIRONMENT.....</b>	<b>17</b>
4.1 Model Generators and Metagenerators .....	18
4.2 Model API & Container Registry .....	21
4.3 Automated Model Generation Prototype.....	22
<b>5. PROOF-OF-CONCEPT IMPLEMENTATION.....</b>	<b>26</b>
5.1 Knowledge Graph.....	26
5.2 Automated Model Generation .....	28
5.3 Service Deployment .....	29
<b>REFERENCES.....</b>	<b>31</b>



## LIST OF FIGURES

Figure 1: Conceptual view of the digital twin lifecycle. ....	8
Figure 2: Overview of the implementation of the technical requirements through the technical framework in relation to the lifecycle phases of the REFORMERS DT. ....	12
Figure 3: Conceptual overview of the runtime environment of a REFORMERS DT service. ....	14
Figure 4: RDP setup for REFORMERS DT with decentralized services. ....	15
Figure 5: Overview of the model development and service deployment workflow for the REFORMERS DT. ....	18
Figure 6: Example generator manifest file. ....	19
Figure 7: Conceptual overview of metagenerators and model generators. ....	20
Figure 8: Sequence diagram of interacting with the Model API for retrieving information about available model generators and models. ....	21
Figure 9: Sequence diagram of interacting with the Model API for creating a new model. ....	22
Figure 10: Overview of the components of the Model API & Container Registry prototype ....	23
Figure 11: Screenshot of the UI for the Model API & Container Registry prototype. ....	24
Figure 12: Example of a model request file (top) and the resulting log messages of the Model API Helper (bottom). ....	25
Figure 13: Power and thrust curves (top left), geographical information (bottom left), and other information is integrated into the knowledge graph (middle). This information can be used to create a configuration file (right) for the wind park simulation. ....	27
Figure 14: Automated model generation workflow including information from the knowledge graph database. ....	28
Figure 15: Overview of the RDP setup for the example REFORMERS DT service. ....	29
Figure 16: Typical forecast results from a wind park simulation as displayed by the dashboard. ....	30

Acronyms	
<b>API</b>	Application Programming Interface
<b>CI/CD</b>	Continuous Integration / Continuous Delivery
<b>DevOps</b>	integration and automation of software development and IT operations
<b>DT</b>	Digital Twin
<b>OCI</b>	Open Container Initiative
<b>OGC</b>	Open Geospatial Consortium
<b>REST</b>	Representational State Transfer
<b>REV</b>	Renewable Energy Valley
<b>UI</b>	User Interface
<b>WP</b>	Work Package



# 1. INTRODUCTION

Digital twins create a virtual counterpart of real-world assets and processes with the help of models and services. The **models** are a digital representation of the corresponding real-world assets and processes and can come in different forms (equation-based, data-driven, etc.). The **services** use (and possibly combine) these models to provide different types of applications (forecasting, monitoring, control, etc.).

The **digital twin lifecycle** introduced in this document provides a structured framework for the development, deployment, and maintenance of the models and services of a digital twin. It describes the typical phases and progression between phases during the development of a digital twin model, from inception to retirement. This helps organizing the development process and eases the task of quality assurance during validation and deployment. In the following, the phases of the digital twin lifecycle are briefly described (see also Figure 1):

- **Phase 1 – Plan:** Define the goal, the assets to model, and the decisions the digital twin should support. Agree on success metrics, data needs, and constraints.
- **Phase 2 – Design:** Choose the modelling approach and interfaces. Integrate a minimum viable prototype of the digital twin model with basic input/output interfaces and data pipelines.
- **Phase 3 – Validate:** Test the digital twin against historical and live data, calibrate parameters, as well as verify reliability and security.
- **Phase 4: Operate:** Deploy the digital twin in the field and monitor data quality, performance, and user impact with clear alerts and logs.
- **Phase 5 – Evolve:** Update the digital twin and if required scale to more assets or sites, while maintaining a version history (and the option for rollbacks). When goals change, decommission cleanly and reuse components.

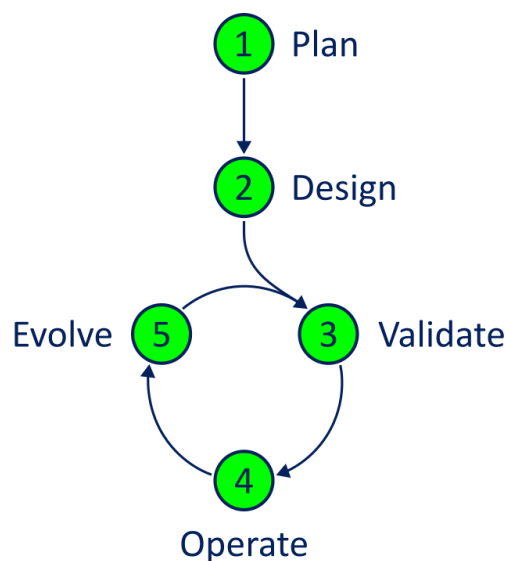


Figure 1: Conceptual view of the digital twin lifecycle.

The REFORMERS Digital Twin (DT) will comprise different types of models (equation-based, data-driven, etc.) for different services, developed by the REFORMERS consortium partners participating in Work Package WP5. This results in an **environment with high diversity**. Even though general best practices for secure software development should be followed [1], [2], this makes it **unfeasible to prescribe one specific approach** to design / implement (phase 2), validate / calibrate (phase 3), and evolve / update (phase 5) a digital twin model.

Instead, this document provides a set of **technical requirements** (see Section 2) for the models and services of the REFORMERS DT. This serves as the common ground for a **technical framework**, enabling a **consistent and streamlined process for the development and operation of the REFORMERS DT across its entire lifecycle**. This technical framework has 3 core components:

1. A **common runtime environment** (see Section 3) provides a system where all models and services of the REFORMERS DT are executed, coordinated, and integrated for specific applications. It includes the underlying infrastructure for running programs, ensures compatibility, and provides essential services.
2. An **automated model generation environment** (see Section 4) serves as the link between the models developed for the REFORMERS DT and its services deployed in the field. It helps with workflow automation and consistent labelling of metadata (model versions, runtime parameters, data sources, etc.), enabling the creation of a living record as part of the REFORMERS DT's lifecycle.
3. A **knowledge graph** provides linked (meta)data for the assets and processes in a Renewable Energy Valley (REV), see Deliverable D5.2 ("*Energy data space for digital twins*"). It acts as the common information source for all models and services of the REFORMERS DT.

This technical framework provides an environment supporting the following roles in the development of the REFORMERS DT:

- **Model developers** are responsible for creating digital twin models. In practice, this typically involves the adaptation and integration of existing algorithms and modelling tools into the common runtime environment.
- **Service developers** use available digital twin models and deploy them as part of a digital twin service. In practice, this involves connecting one or more digital twin models to real-time data sources and REV assets as well as linking them with each other to enable a certain use case.

The same person may be both a model developer and a service developer. However, this distinction of roles underpins the difference in model development and service deployment as part of the digital twin lifecycle.

The following sections provide details on the technical requirements (Section 2), the common runtime environment (Section 3), and the automated model generation environment (Section 4). Section 5 showcases a proof-of-concept implementation of a



REFORMERS DT service, taking full advantage of the technical framework described in this document to integrate of an existing forecasting algorithm.

All the software and examples mentioned in this report are publicly available under open-source licenses. Links to the software repositories are provided in the corresponding sections.



**Funded by  
the European Union**

**Project funded by**



Schweizerische Eidgenossenschaft  
Confédération suisse  
Confederazione Svizzera  
Confederaziun svizra

Swiss Confederation

Federal Department of Economic Affairs,  
Education and Research EAER  
**State Secretariat for Education,  
Research and Innovation SERI**

## 2. DIGITAL TWIN TECHNICAL REQUIREMENTS

The main technical requirements for the REFORMERS DT stem from the need to rapidly develop, deploy, and test innovative services in real-world environments. These technical requirements are essential to ensure interoperability and scalability when integrating heterogeneous models, data sources, protocols, and deployment contexts as part of the REFORMER DT.

The requirements can be grouped into design principles, functional capabilities, and deployment needs:

### Design Principles:

- **Modularization:** adopt a microservice architecture where each module focuses on a single, clearly defined task (e.g., data retrieval, forecasting, control)
- **Use of Mature Components:** prioritize the integration of well-tested, open-source components to enhance reliability and reduce development time
- **Reusability & Flexibility:** design interfaces and components to be reusable across different REVs, provide default configurations to ease system integration
- **Security:** include state-of-the-art security measures by design (e.g., dynamic reverse proxy management, secure web endpoint routing, encryption, network segmentation)

### Functional Capabilities:

- **Data Acquisition:** use polling-based and event-driven data retrieval (e.g., from smart meters or weather services)
- **Data Processing:** allow real-time data exchange via in-memory databases and long-term storage via persistent databases
- **Scalability:** allow scaling to a large number of assets and high data volumes
- **Algorithm Integration:** support integration of a wide variety of algorithms and real-time controllers
- **Communication Protocols:** enable handling of diverse protocols (e.g., Modbus, REST, MQTT) to communicate with field devices and external systems
- **Monitoring:** use fine-grained system monitoring and alerting to ensure stability (also during prototype testing)

### Deployment:

- **Knowledge Graph Access:** enable access to a REV's knowledge graph as a common information source for REFORMERS DT models, especially for automation of service deployment
- **Containerization:** package REFORMERS DT models and their dependencies in virtual containers that can run on any platform, enabling them to run in a variety of locations (e.g., on-premises, public / private cloud)



- **Workflow Automation:** use container orchestration (e.g., Docker, Podman) and DevOps integration (e.g., GitHub Actions, GitLab CI/CD) for runtime management, automated deployment, and backups

In summary, the technical requirements for the REFORMERS DT emphasize modularity, reliability, interoperability, and operational efficiency, ensuring that novel energy-related services can be deployed and tested quickly across various real-world setups.

These technical requirements have been implemented through the technical framework introduced in Sections 3 and 4. Figure 2 gives an overview of how the technical requirements and their implementation relate to the lifecycle phases of the REFORMERS DT. It shows that the common runtime environment provides the foundation for the design, validation, and operation of the REFORMERS DT models. The automated model generation environment facilitates the models' deployment as part of the REFORMERS DT services. The knowledge graph is primarily used during runtime of a REFORMERS DT model.

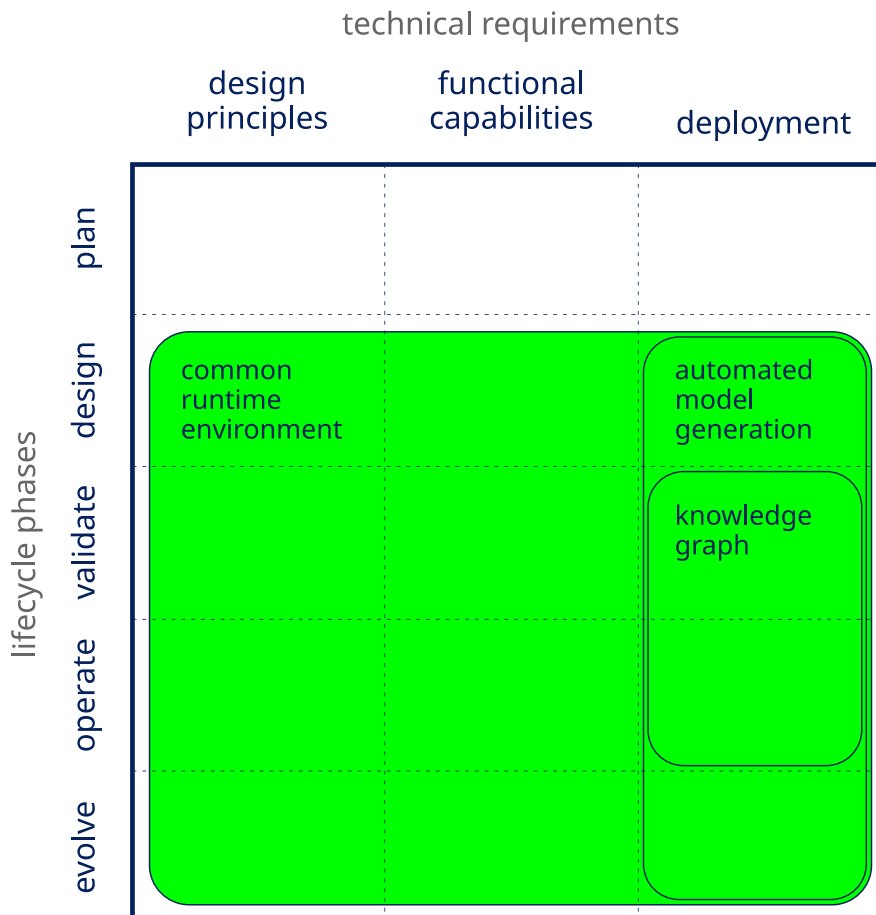


Figure 2: Overview of the implementation of the technical requirements through the technical framework in relation to the lifecycle phases of the REFORMERS DT.

### 3. RUNTIME ENVIRONMENT FOR MODELS AND SERVICES

The Rapid Deployment Platform (RDP) has been selected as runtime environment for the REFORMERS DT. Developed by AIT, it is a modular software stack that bundles data connectors, storage, messaging, and applications in containers so that energy-related services (digital twins, energy management systems, etc.) can be built, tested, deployed, and operated reliably across different environments. The RDP core components are publicly available under an open-source license<sup>1</sup>.

#### 3.1 General RDP Architecture

The RDP architecture is in agreement with the requirements specified in Section 2. In the following, a brief overview of its main features is provided.

The RDP splits the solution into small modules, each doing a single, clearly defined task. New digital twin services can be assembled from these **building blocks** instead of being written from scratch. This shortens the time from concept to a working prototype and makes revisions quick when the model or inputs change.

Digital twin services need measurements, forecasts, and control channels. The RDP provides a **connection to many data sources** as it already speaks common field and web protocols and offers both polling and event-driven connectors. This lets a digital twin service connect to meters, inverters, weather services, and enterprise systems without site-specific code.

The RDP provides reliable **real-time data flow with long-term storage**. A fast in-memory data layer handles the live state and predictions used by the digital twin models. A separate time-series database stores history for training and evaluation. The two layers are synchronized automatically, so real-time performance and analytics do not interfere with each other. Prior deployments show high sample rates and very large data volumes can be handled, which supports growth in assets and sites without major redesign.

The RDP supports **hybrid execution patterns**. Digital twin services often mix periodic simulations, continuous state estimation, and event-driven controls. The RDP accommodates different timing and interaction schemes within one installation, so operators can combine forecasting, optimization, and device control in a coherent flow.

The RDP promotes **security-by-design**, as reverse proxies, encryption, and fine-grained network segmentation are part of the baseline. Role-based access to data and services helps sharing access to a digital twin service across partners while protecting sensitive information.

The RDP approach favours mature open components with thin custom adapters, promoting **vendor-neutrality and reusability**. Interfaces are unified where possible, yet flexible when

<sup>1</sup> AIT Rapid Deployment Platform: <https://ait-rdp.github.io/>



needed. This reduces vendor lock-in and allows reusing connectors and data models across different twins and projects.

The RDP architecture relies on **containerization**. Containerized services allow the same digital twin to run on a laptop, an industrial PC, or a server cluster. Moreover, RDP applications are typically integrated into **DevOps workflows** (GitHub Actions, GitLab CI/CD, etc.) for continuous deployment and close operations feedback to shorten development cycles.

In summary, the RDP provides the REFORMERS DT with a ready-made “operating system” for data, computation, and control. It speeds up deployment, improves reliability, and makes it practical to run the same digital twin service across sites and scales while focusing engineering effort on the digital twin’s logic rather than the plumbing.

### 3.2 REFORMERS DT Services and Models Architecture

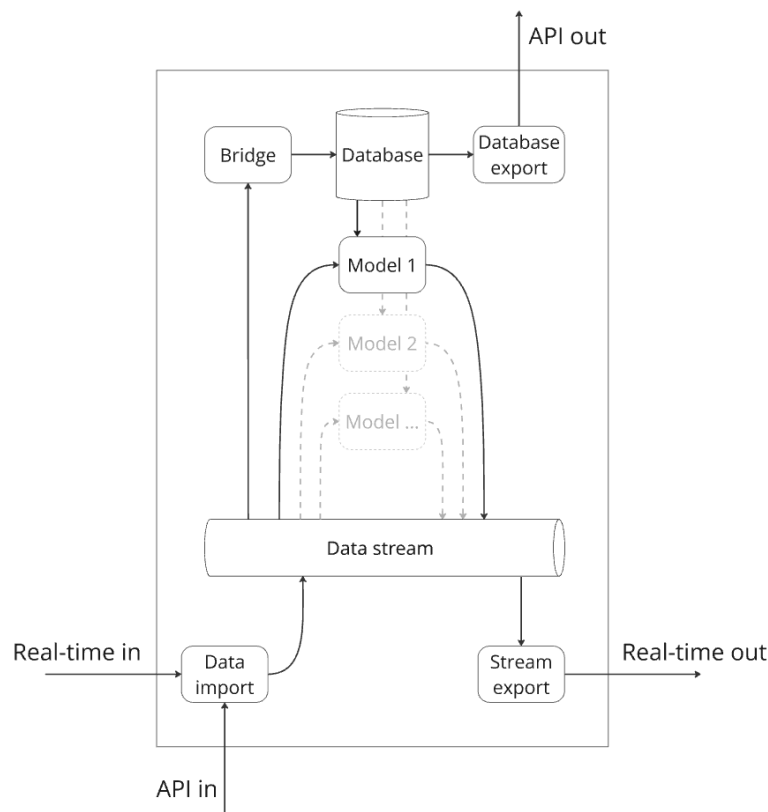


Figure 3: Conceptual overview of the runtime environment of a REFORMERS DT service.

Figure 3 shows the conceptual overview of the RDP architecture applied to REFORMERS DT services:

- Data from REV assets (e.g., sensors) and external sources (e.g., weather forecast services) is made available via a fast in-memory **data stream**.

- Digital twin **models** operate primarily on the data available through this data stream. Data generated by the models is written back to the data stream and optionally sent to REV assets (e.g., controller setpoints).
- Data can be selectively stored in a time-series **database**. This database is intended for long-term storage. Its content can be used to provide historical data for the models or for monitoring.
- Data may also be **exported** and made available through **dedicated APIs** (e.g., REST API, OGC Process API) for visualization or further processing by external services.

Within the RDP architecture, **REFORMERS DT models** are modules like any other. As such, they **must be implemented as RDP-compliant executable containers**, reading and writing data from the data stream.

### 3.3 Decentralized REFORMERS DT Setups

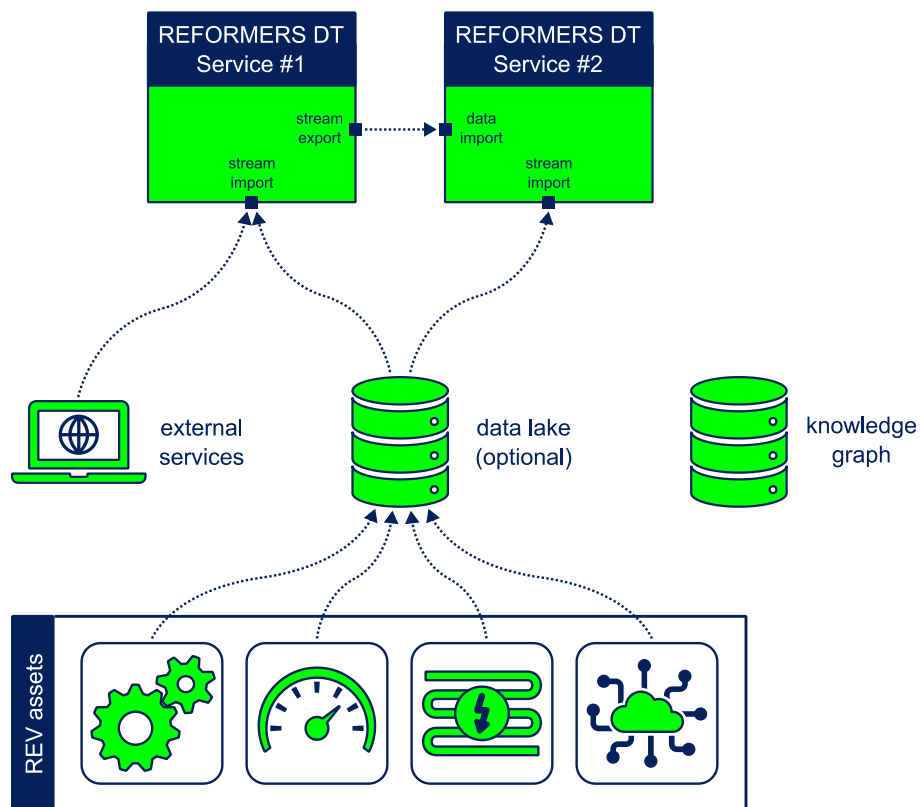


Figure 4: RDP setup for REFORMERS DT with decentralized services.

The RDP allows the deployment of REFORMERS DT setups in a decentralized manner, with services running at different locations. Figure 4 shows an example of 2 separate REFORMERS DT services. Each of the services receives data from the REV assets and/or external services. Data from the REV assets can be cached in a storage (e.g., a data lake)

or directly streamed to the services. In addition, services may also exchange data directly between each other.

Such a setup is particularly useful for the following use cases:

- **Decentralized Service Deployment:** Multiple digital twin services may depend on each other yet be deployed at different locations. In the case of REFORMERS, partners may develop their own digital twin services and choose to host them on their own infrastructure. For instance, digital twin service #1 may control a REV asset, while depending on forecasts for prices and PV production from digital twin service #2.
- **Sensitive Data Processing:** A decentralized setup with individual digital twin services processing data independently can help with the handling of sensitive data. For example, digital service #1 could process data to which service #2 is not allowed access. For instance, this data could be subject to a non-disclosure agreement between the data owners and the operators of service #1. The resulting output, however, could be not affected by this restriction (e.g., due to sufficient anonymization or aggregation) and may be shared with service #2 for further processing.



Funded by  
the European Union

Project funded by



Schweizerische Eidgenossenschaft  
Confédération suisse  
Confederazione Svizzera  
Confederaziun svizra

Swiss Confederation

Federal Department of Economic Affairs,  
Education and Research EAER  
State Secretariat for Education,  
Research and Innovation SERI

## 4. AUTOMATED MODEL GENERATION ENVIRONMENT

The RDP serves as the common runtime environment for the REFORMER DT. It provides a common technical baseline for the design and validation of models as well as the operation of services. However, even though the RDP architecture promotes the use of DevOps workflows, it does not provide, by itself, tools for workflow automation and consistent labelling of metadata (model versions, runtime parameters, data sources, etc.). Nevertheless, this is crucial for enabling the creation of a living record as part of the REFORMERS DT's lifecycle.

To this end, an **automated model generation environment** has been developed in Work Package WP5, which serves as the link between the models developed for the REFORMERS DT and its services deployed in the field. It comprises the following components:

- The **model generators** automate the creation of executable containers (i.e., the models in the sense of Section 3.2) from model source code. Model generators are themselves containers and can be easily integrated into automated DevOps workflows.
- The **Model API & Container Registry** is a central repository to store and retrieve model generators and models. It provides a REST-based interface to retrieve information about model generators and models as well as request the creation of new models.

Figure 5 shows an overview of the overall workflow for the development and deployment of the REFORMERS DT. The automated model generation environment supports the digital twin lifecycle primarily in two ways:

- Model developers create source code during phase 2 (design), phase 3 (validate), and phase 5 (evolve). Once they are satisfied with their results, they can publish a corresponding model generator to the Model API & Container Registry. The **generators are published as container images, together with metadata about their intended use and provenance**. This process can be automated as part of a DevOps setup, e.g., publishing a new model generator whenever a new version has been released.
- Service developers create service definitions and configurations during phase 4 (operate). This should include a DevOps setup for the automated deployment of the service, which relies on the generators published via the Model API & Container Registry to create (if needed) and retrieve models. **New models are published as container images, together with metadata about their intended use and provenance**. Upon retrieval for use in a service deployment, a model's metadata can be checked to verify its correct use.

Metadata for generators and models is stored as labels in their corresponding container images. This metadata contains information about their provenance. For model generators, this includes their name and version, creation date, mandatory generator configuration, and



model build parameters (see Section 4.1 for details). For models, this includes their name and version, creation date, the name and version of the generator used to create them, the generation parameters, and the mandatory and optional runtime parameters (see Section 4.2 for details). This **metadata provides the provenance required for traceability** of digital twin models of the REFORMERS DT, creating a living record as part of its lifecycle.

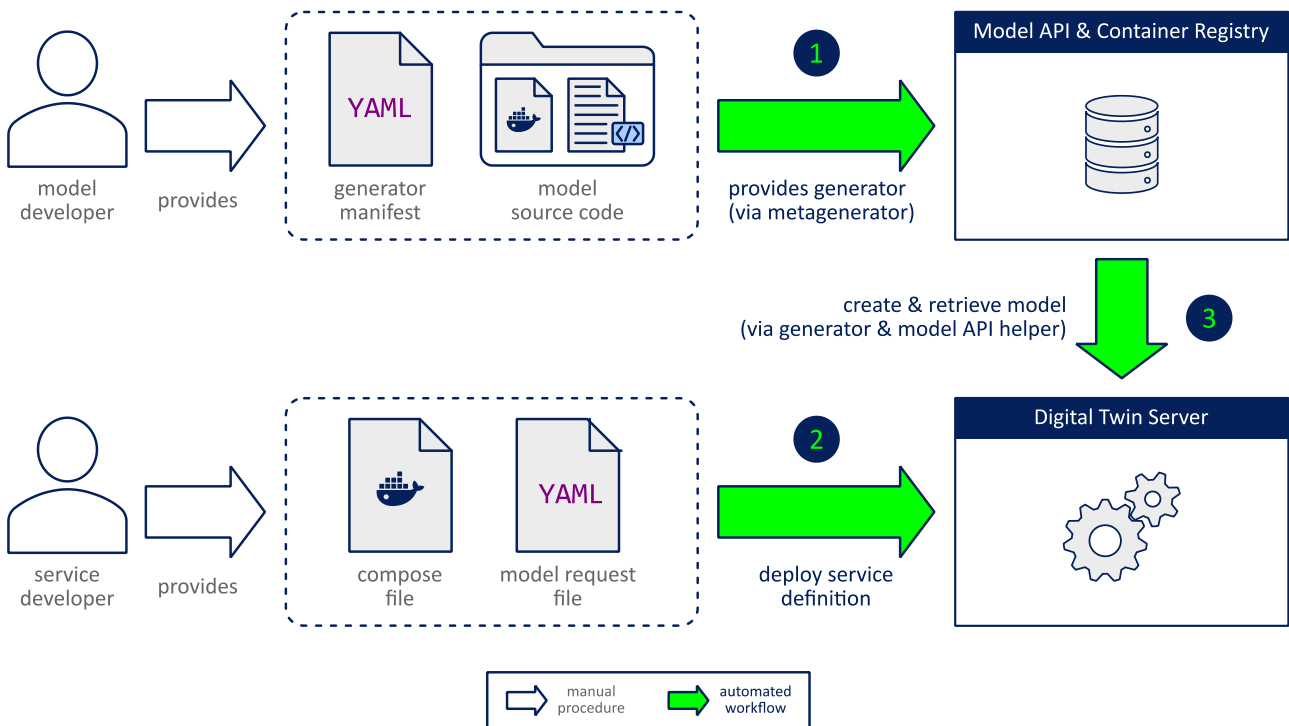


Figure 5: Overview of the model development and service deployment workflow for the REFORMERS DT.

### 4.1 Model Generators and Metagenerators

To comply with the architecture of the common runtime environment of the REFORMERS DT (i.e., the RDP), its models are containerized and published as container images. Model generators relieve model developers of the following tasks:

- **Model Containerization:** The model generation process is automated via the Model API & Container Registry. To this end, the generators are themselves containerized applications. This means that the model container images are built within another running container (i.e., the generator), which is achieved by using the kaniko build tool<sup>2</sup>. Usage of kaniko is not trivial, and model generators provide an easy-to-use wrapper around it.
- **Metadata Creation:** For the purpose of the Model API & Container Registry, all metadata is stored as labels within the container images. However, data stored in such labels can be arbitrary. Model generators make sure that the metadata is stored

<sup>2</sup> Kaniko: <https://github.com/chainguard-dev/kaniko>

consistently across all created container images, making the generation process less error-prone.

Model developers can create generators from their model's source code with the help of the so-called metagenerator<sup>3</sup>. The metagenerator is a user-friendly tool that takes the following inputs:

- the **model source code** for the specific model (including its Dockerfile for containerization),
- a **generator manifest** file, and
- the **registry credentials** for publishing the new generator to the Model API & Container Registry.

The generator manifest is a YAML-formatted file that contains the meta-information required for the build process (see Figure 6 for an example):

- generator name and version (tag),
- mandatory parameters for the generator configuration, which typically stay the same for different versions of the same model,
- a list of the runtime parameters of the models it generates, incl. a short description and default values, and
- optional build parameters.

```
1  ---
2  example-generator:
3    version: v0
4    config:
5      # URL to generator registry
6      GENERATOR_REGISTRY: reformers-dev.ait.ac.at:8082
7      # URL to model registry
8      MODEL_REGISTRY: reformers-dev.ait.ac.at:8083
9      # Dockerfile for the model build process (relative to model source directory)
10     MODEL_DOCKERFILE: Dockerfile_model
11   parameters:
12     CONFIG_FILE:
13       info: path to config file with default values
14       default: /config/config.yml
15     GRID_DATA:
16       info: path to grid data
17       default: /grid_data/grid.json
18   build:
19     cache:
20       - python:3.13
```

Figure 6: Example generator manifest file.

<sup>3</sup> REFORMERS Metagenerator for DT Model Generators:  
<https://github.com/REFORMERS-EnergyValleys/reformers-dt-metagenerator>



The metagenerator is itself a containerized application, making it easy for model developers to use in a DevOps setup. For instance, the metagenerator could create a new model generator and publish it via the Model API & Container Registry every time the mode source code is tagged with a new version number.

A detailed view of the workflow for metagenerators and model generators is depicted in Figure 7, providing a more comprehensive view of how they are used as part of the processes shown in Figure 5. A stand-alone example for using the metagenerator to create a model generator from a generator manifest file and model source code is available on the REFORMERS code repository<sup>4</sup>.

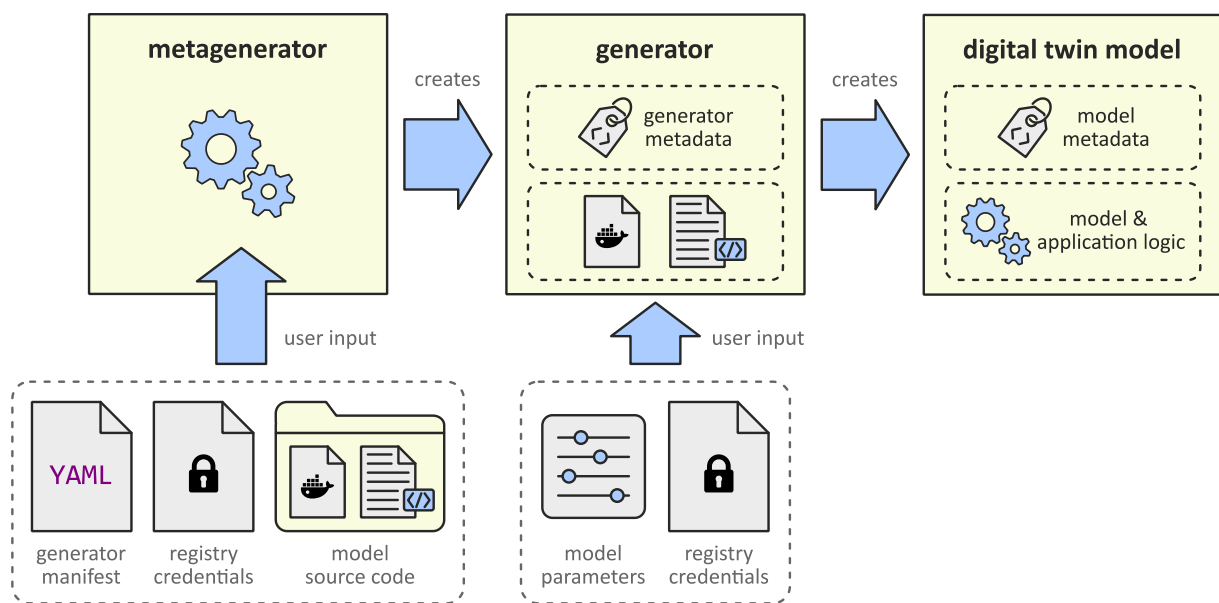


Figure 7: Conceptual overview of metagenerators and model generators.

<sup>4</sup> Example model generator for the REFORMERS DT:  
<https://github.com/REFORMERS-EnergyValleys/example-model-generator>

## 4.2 Model API & Container Registry

The Model API & Container Registry is a central component in the technical framework enabling the lifecycle of the REFORMERS DT. It has 3 main tasks:

- **Centralized Repository:** The container images of all generators and models are stored in dedicated OCI-compliant<sup>5</sup> container image registries.
- **Metadata Provisioning:** All metadata for generators and models is stored as labels in the associated container images. The Model API provides easy access to this information. Figure 8 shows the sequence diagram of interacting with the Model API for retrieving information about available model generators and models.
- **Model Creation:** Upon request, the Model API can create (and store) new models with the help of the available model generators. Figure 9 shows the sequence diagram of interacting with the Model API for creating a new model.

The Model API provides a REST-based web interface, which can be easily integrated into (automated) workflows. The specification of the interface is publicly available<sup>6</sup>.

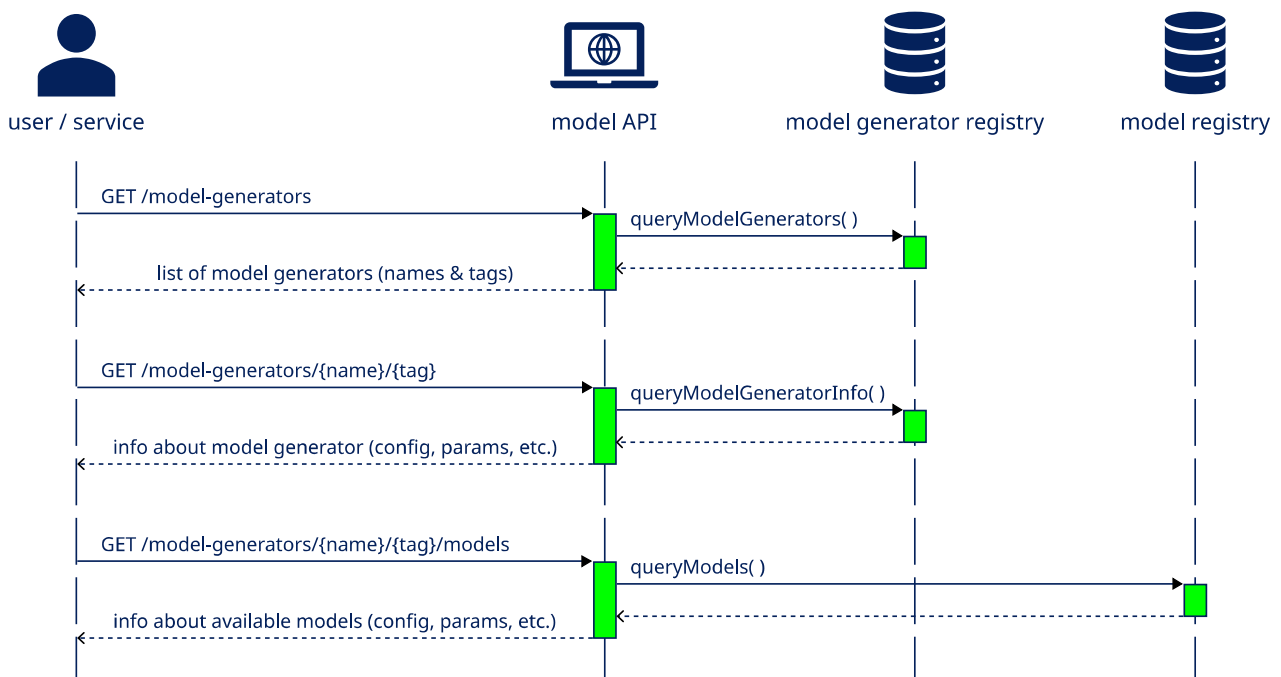


Figure 8: Sequence diagram of interacting with the Model API for retrieving information about available model generators and models.

<sup>5</sup> Open Container Initiative Distribution Specification: <https://github.com/opencontainers/distribution-spec>

<sup>6</sup> API Documentation for REFORMERS Digital Twin Model API: <https://reformers-energyvalleys.github.io/reformers-dt-model-api-specs/>



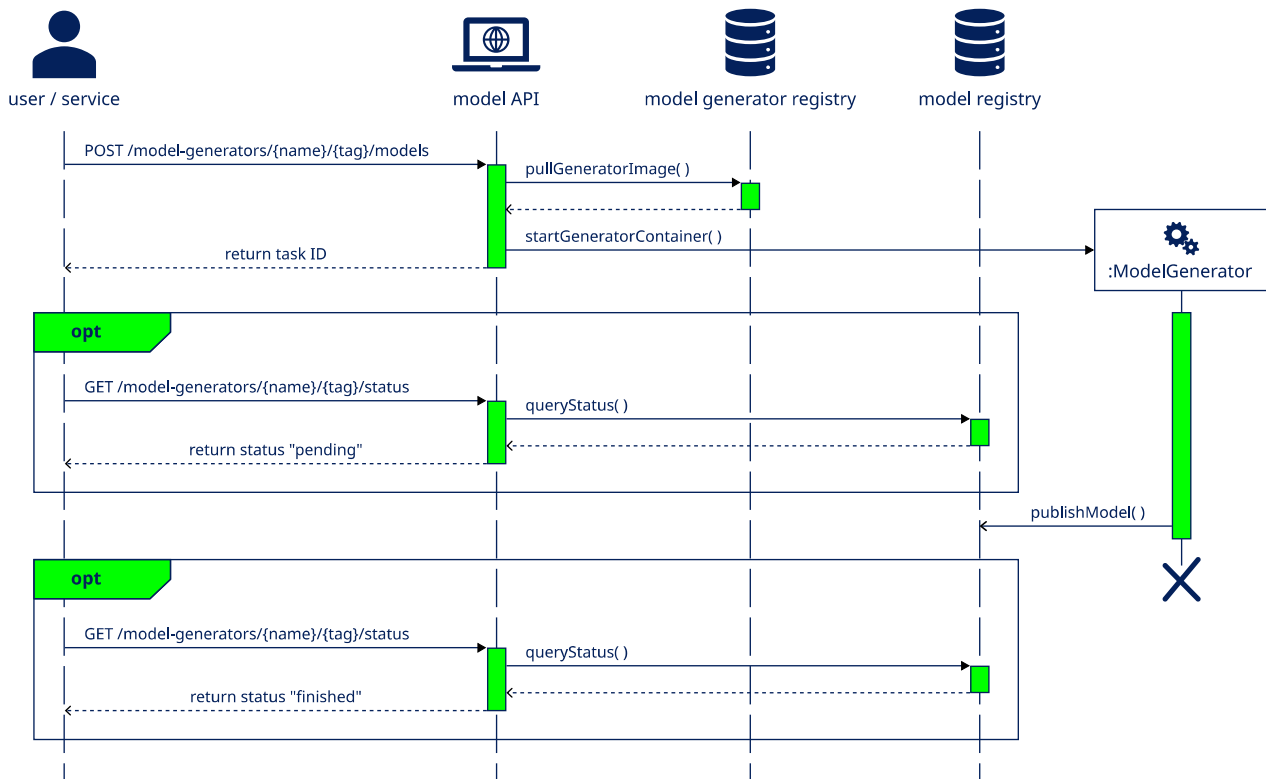


Figure 9: Sequence diagram of interacting with the Model API for creating a new model.

### 4.3 Automated Model Generation Prototype

A prototype of the Model API & Container Registry and related components has been developed as proof-of-concept for the automated model generation framework<sup>7</sup>. For this prototype, the following design choices have guided the implementation:

- **Minimalistic:** The prototype design follows a minimalist approach that attempts to use as few components as possible. For instance, the choice to store metadata as labels in container images means that no additional database for metadata is required.
- **Software Quality:** Available high-quality software is used wherever possible. For instance, server stubs and API client implementations have been generated from their specifications using OpenAPI generators<sup>8</sup>.

In addition, the following implementation choices have been followed:

- **Open Source:** All software developed for the prototype is released under an open-source license (MIT license) and is publicly available.

<sup>7</sup> REFORMERS DT Model API & Container Registry: <https://github.com/REFORMERS-EnergyValleys/reformers-dt-model-api>

<sup>8</sup> OpenAPI Generator: <https://openapi-generator.tech/>

- **Open Standards:** All APIs have been specified using the OpenAPI standard<sup>9</sup> and are publicly available<sup>10</sup>.

Figure 10 shows an overview of the components of the Model API & Container Registry prototype:

- For the **Container Registry**, the Sonatype Nexus Repository Community Edition<sup>11</sup> is used. On top of a high-quality solution for hosting container image registries, it also provides an interface for advanced searches of repository content. This is an important feature for extracting metadata for generators and models. The prototype implements two separate container image registries, one for the generator images and one for the model images, respectively.
- The **Model API** is hosted with the help of a dedicated **server**<sup>12</sup>. Its implementation is based on stubs generated from its API specification and includes a web-based **user interface** (see Figure 11). The metadata stored in the container registry is accessed via a dedicated **client**<sup>13</sup>, which connects to the container image registries and the search interface provided through the Sonatype Nexus Repository. The client is entirely generated from its API specification.
- A local Docker daemon is used to run generators and create new models.

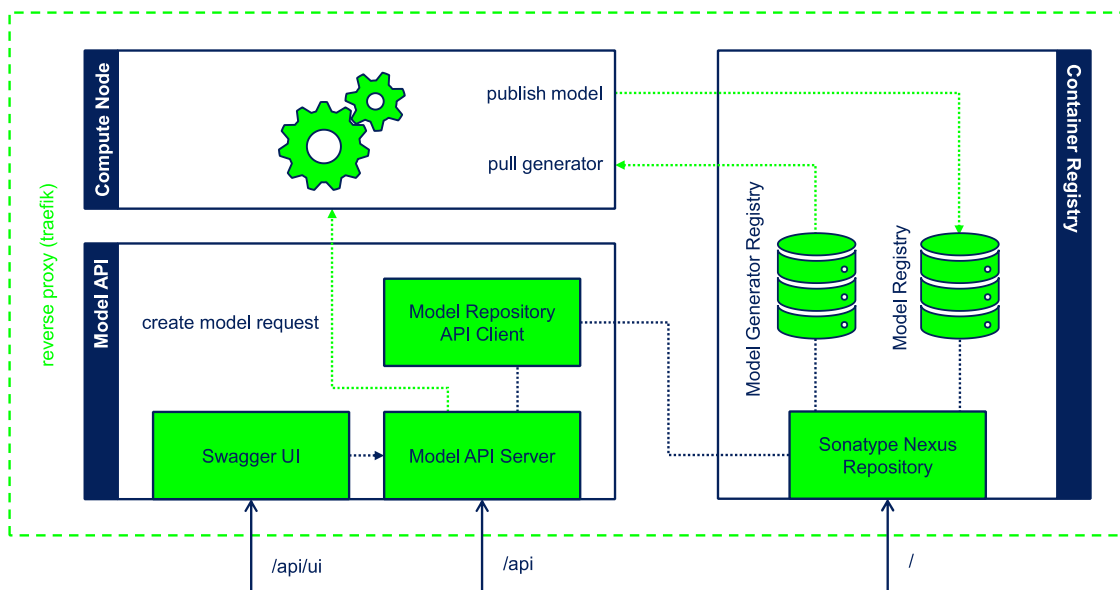


Figure 10: Overview of the components of the Model API & Container Registry prototype

<sup>9</sup> OpenAPI Specification: <https://www.openapis.org/>

<sup>10</sup> OpenAPI Specification for REFORMERS DT Model API: <https://github.com/REFORMERS-EnergyValleys/reformers-dt-model-api-specs>

<sup>11</sup> Sonatype Nexus Repository Community Edition: <https://help.sonatype.com/en/ce-onboarding.html>

<sup>12</sup> REFORMERS DT Model API Server: <https://github.com/REFORMERS-EnergyValleys/reformers-dt-model-api-server>

<sup>13</sup> REFORMERS DT Model Repository Client: <https://github.com/REFORMERS-EnergyValleys/reformers-dt-model-repository-client>

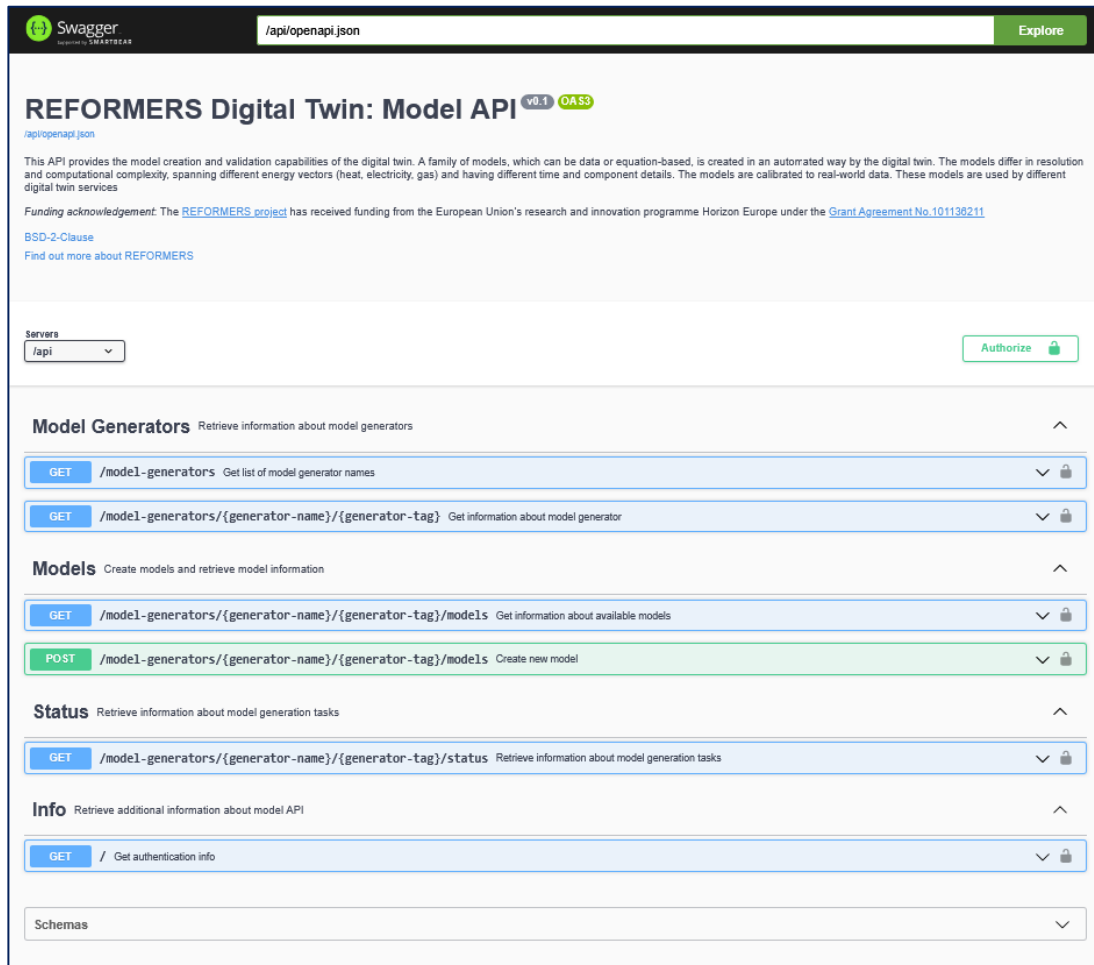


Figure 11: Screenshot of the UI for the Model API & Container Registry prototype.

To ease the use of the Model API & Container Registry as part of automated DevOps workflows for REFORMERS DT service deployments, the so-called **Model API Helper** is provided<sup>14</sup>. It checks the availability of a specified model and generates it in case the model is not yet available. Service definitions need to include a **model request file** as input to the Model API Helper, which in turn creates an environment variable file for configuring the model at runtime. See Figure 12 for an example.

The Model API helper is packaged into a light-weight container and is available via the GitHub container registry. This container can be integrated with little effort into existing DevOps workflows (e.g., GitHub Actions, GitLab CI/CD). An example setup is provided on the REFORMERS code repository<sup>15</sup>.

<sup>14</sup> REFORMERS DT Model API Helper:

<https://github.com/REFORMERS-EnergyValleys/reformers-dt-model-api-helper>

<sup>15</sup> Example RDP Setup for the REFORMERS Digital Twin:

<https://github.com/REFORMERS-EnergyValleys/example-rdp-setup/tree/with-model-api>



```

1  generator_name: example-generator
2  generator_tag: v0
3  model_name: grid-sim
4  model_tag: latest
5  force_generation: false
6  generation_parameters:
7    OUTPUT_STREAM_BASE: reformers.grid_sim_test.latest.results
8    INPUT_STREAM: reformers.metering_data.latest.DUMMY1
9  parameters:
10  EXTRA_CONFIG: /config/extra-config.yml
  
```



```

✔ Model 'grid-sim' with version 'latest' found
! Model 'grid-sim' with version 'latest' will be generated
✔ All generation parameters are valid
! Started model generation task with ID=Z3JpZC1zaW06bGF0ZXN0jE3NTU2MDMzNzAuNzA1NDk5
⌚ Current status: pending - checking again in 15s ...
⌚ Current status: pending - checking again in 15s ...
⌚ Current status: pending - checking again in 15s ...
⌚ Current status: pending - checking again in 15s ...
✔ Task Z3JpZC1zaW06bGF0ZXN0jE3NTU2MDMzNzAuNzA1NDk5 finished
✔ All checks passed.
✔ File generated: model.env
  
```

Figure 12: Example of a model request file (top) and the resulting log messages of the Model API Helper (bottom).

## 5. PROOF-OF-CONCEPT IMPLEMENTATION

This section provides an example that leverages the technical framework presented in Sections 3 and 4 as well as the knowledge graph described in Deliverable D5.2. The example showcases how an existing algorithm can be integrated as part of the REFORMERS DT. This integration comprises 3 steps:

1. gather information about the REV assets and make it available via a knowledge graph database (see Section 5.1),
2. integrate the forecasting algorithm into an RDP module and provide a dedicated generator to create an executable REFORMERS DT model on demand (see Section 5.2), and
3. provide a REFORMERS DT service whose deployment is automated via a DevOps workflow (see Section 5.3).

For the purpose of this example, the EOLICA<sup>16</sup> package for forecasting the power generation of wind turbines is used. It relies on wind forecasts from a public weather forecast provider to simulate the behaviour of arbitrary wind parks and generate forecasts based on the simulation results. EOLICA is a Python-based application that was not specifically developed to be deployed via the REFORMERS DT technical framework.

### 5.1 Knowledge Graph

The example provides a fully instantiated knowledge graph representing a wind park, its wind resource characterization, and several wind turbines. For each wind turbine, the geometry, technical specifications, and geolocation are provided. Scenario-level links integrate all components into one scenario. This information is sufficient to generate a configuration file for the EOLICA package (cp. Section 5.2); see Figure 14 for a conceptual overview.

The knowledge representation in the example makes use of:

- core concepts from the **DigiCities Core Ontology** (e.g., `Scenario`, `ComponentLink`), and
- domain concepts from **REFORMERS-specific extension** of the ontology (e.g., `WindTurbine`, `WindTurbineType`, `RotorDiameter`, `PowerCurve`, etc.).

The knowledge graph example comprises the following entities:

- The **main scenario definition** (`wind_forecasting:BaselineAlkmaar`) represents the wind-forecasting baseline scenario and acts as the root node linking all components (wind park, turbines, power curves, parameters). It is typed as `dici_core:Scenario`.

<sup>16</sup> EOLICA: <https://github.com/REFORMERS-EnergyValleys/eolica>



- The **wind park definition** (`windpark_alkmaar:WindparkAlkmaar`) includes a roughness specification (surface roughness at site) and a collection of linked wind turbine components. It is typed as `dici_reformers:GlobalWindAtlasSite`.
- The **individual wind turbines** (`wind_forecasting:wind_turbine:Turbine1`, etc.) are described by their geospatial attributes (latitude, longitude, altitude), hub height, rotor diameter, rated power, and turbine type. Each wind turbine is typed as `dici_reformers:WindTurbine`.
- Each wind turbine is linked to a **turbine type** (`wind_turbine:Type:Enercon66_1p8MW`, etc.) that characterizes its power curve (power output as function of wind speed) and thrust curve (thrust coefficient as function of wind speed). Turbine types are typed as `dici_reformers:WindTurbineType`.

The knowledge graph and its implementation as a knowledge graph database are publicly available under an open-source license<sup>17</sup>. This also includes a notebook showing how to query the database to retrieve the information relevant for this example.

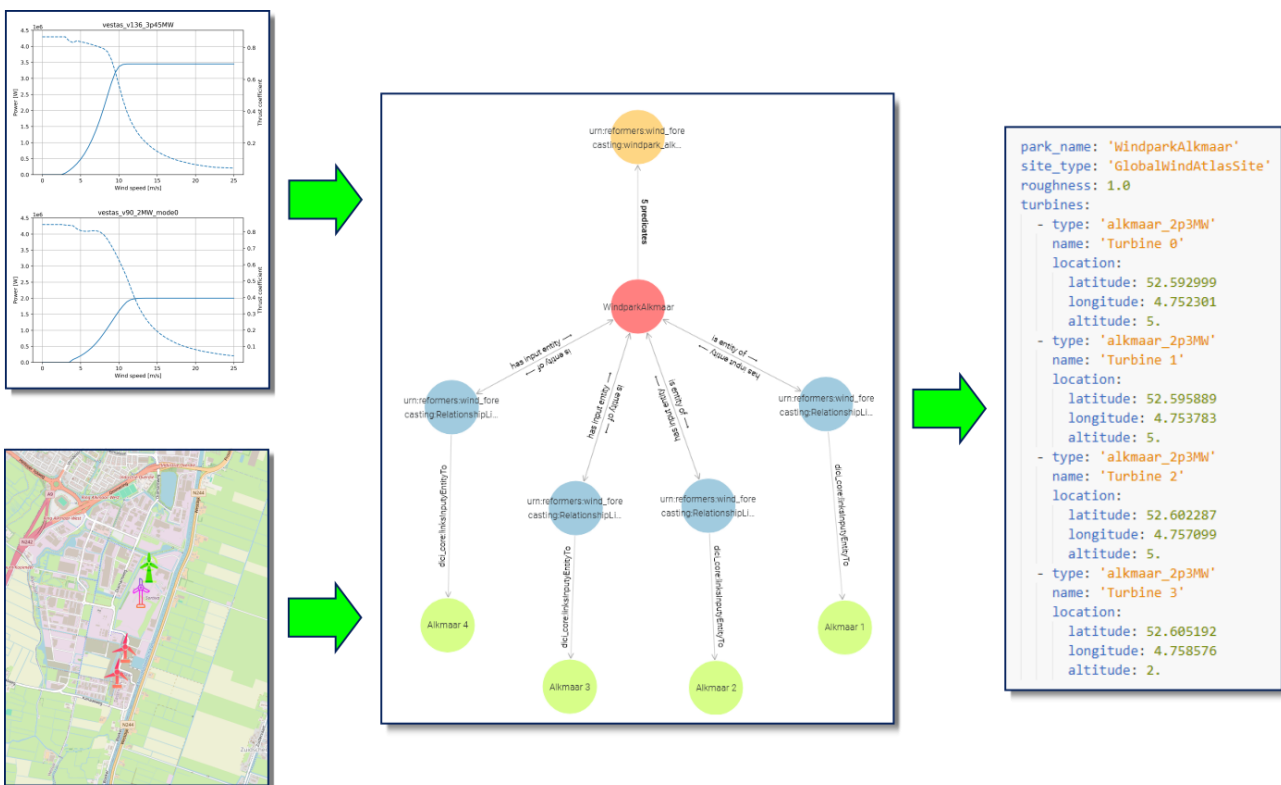


Figure 13: Power and thrust curves (top left), geographical information (bottom left), and other information is integrated into the knowledge graph (middle). This information can be used to create a configuration file (right) for the wind park simulation.

<sup>17</sup> Example knowledge graph for the REFORMERS DT: <https://github.com/REFORMERS-EnergyValleys/example-reformers-knowledge-graph>

## 5.2 Automated Model Generation

The EOLICA tool has been developed as a stand-alone Python package, independently of the REFORMERS DT technical framework. To integrate it into the REFORMERS DT technical framework, a simple wrapper has been developed on top of EOLICA’s core code:

- At startup, the tool configuration is read from a local configuration file or generated from information retrieved from the knowledge graph database.
- Input data (weather forecasts) is read from the data stream, and results (wind power generation forecasts) are written back to the data stream.
- Simulations are executed periodically using a scheduler.
- Instructions for containerizing the application (Dockerfile) into an executable REFORMERS DT model are provided.
- A generator manifest file (cp. Section 4.1) is provided.

The resulting REFORMERS DT wind forecasting model is publicly available under an open-source license<sup>18</sup>.

In addition to the model source code itself, a DevOps configuration (for GitLab CI/CD) has been added. This implements the automated creation of a new model generator (using the generator manifest file and metagenerator) every time a new version of the model source code is released (i.e., the code is tagged with a new version). Together with the knowledge graph database (cp. Section 5.1) and the Model API & Container Registry prototype (cp. Section 4.3), this demonstrates the automated model generation workflow shown in Figure 14.

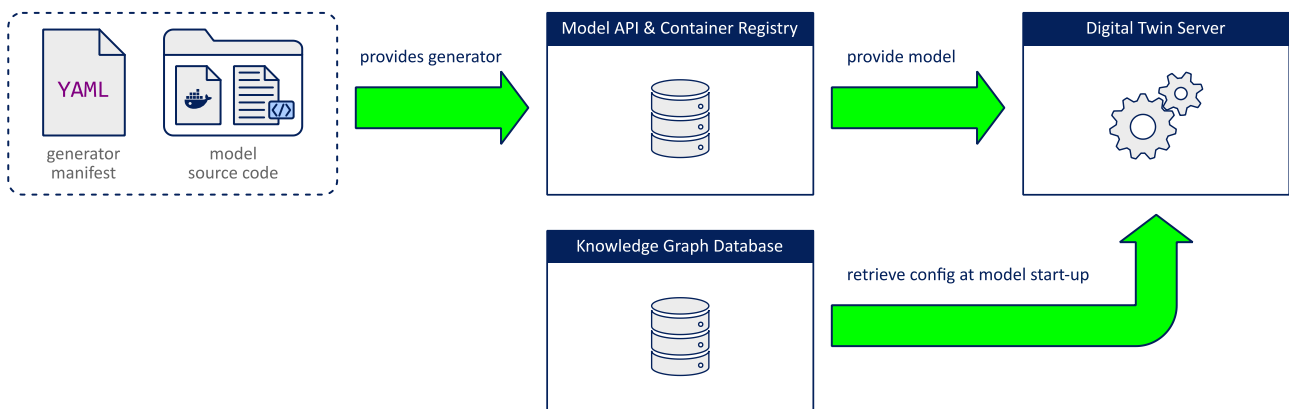


Figure 14: Automated model generation workflow including information from the knowledge graph database.

<sup>18</sup> REFORMERS DT Wind Generation Forecasting Model:  
<https://github.com/REFORMERS-EnergyValleys/reformers-dt-wind-gen-fc-model>

### 5.3 Service Deployment

The executable model described in Section 5.2 is deployed as part of a REFORMERS DT service for forecasting wind power generation. An overview of the RDP setup used for the service is shown in Figure 15. The model itself is augmented with a data crawler for retrieving forecast data from a public weather forecast service as well as a dashboard for visualization.

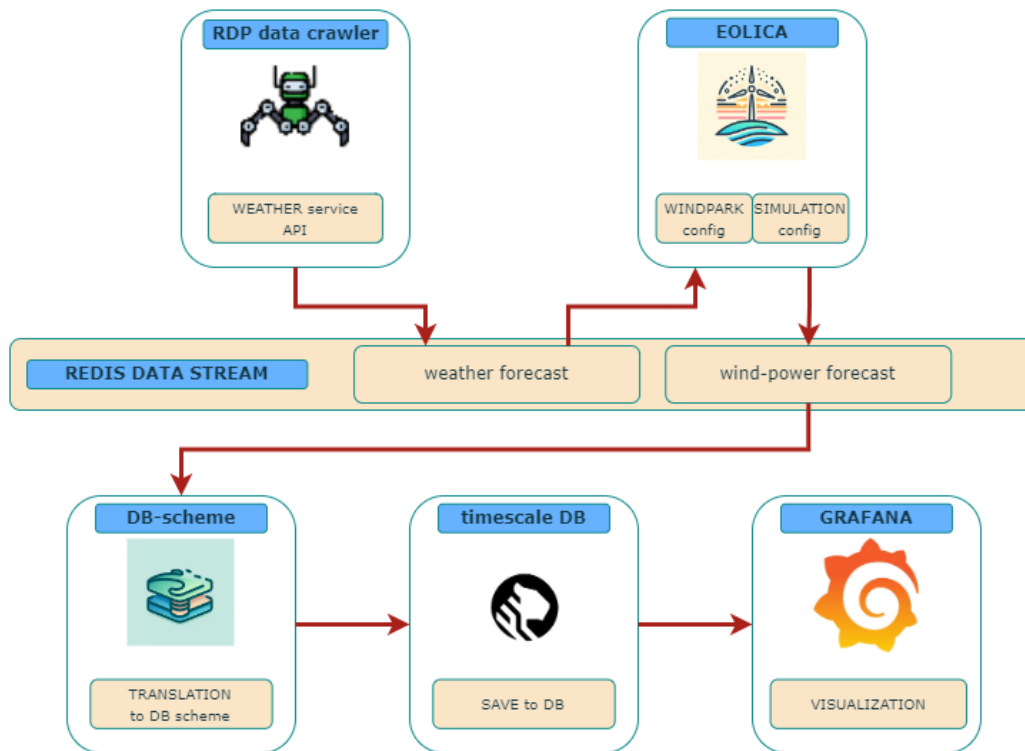


Figure 15: Overview of the RDP setup for the example REFORMERS DT service.

The definition and configuration of the service is publicly available under an open-source license<sup>19</sup>. In addition, a DevOps configuration (for GitLab CI/CD) has been added. This implements the automated deployment of the service (using the Model API helper and a model request file) every time the definition of the service changes.

Figure 16 shows typical results for the wind power generation forecast service as displayed by the service. Retrieval of these results via additional interfaces (e.g., REST API, OGC Process API) or the data export stream can be added with little effort, thanks to the modular approach of the REFORMERS DT runtime architecture.

<sup>19</sup> REFORMERS DT Wind Forecasting Service:  
<https://github.com/REFORMERS-EnergyValleys/reformers-dt-wind-gen-fc-service>



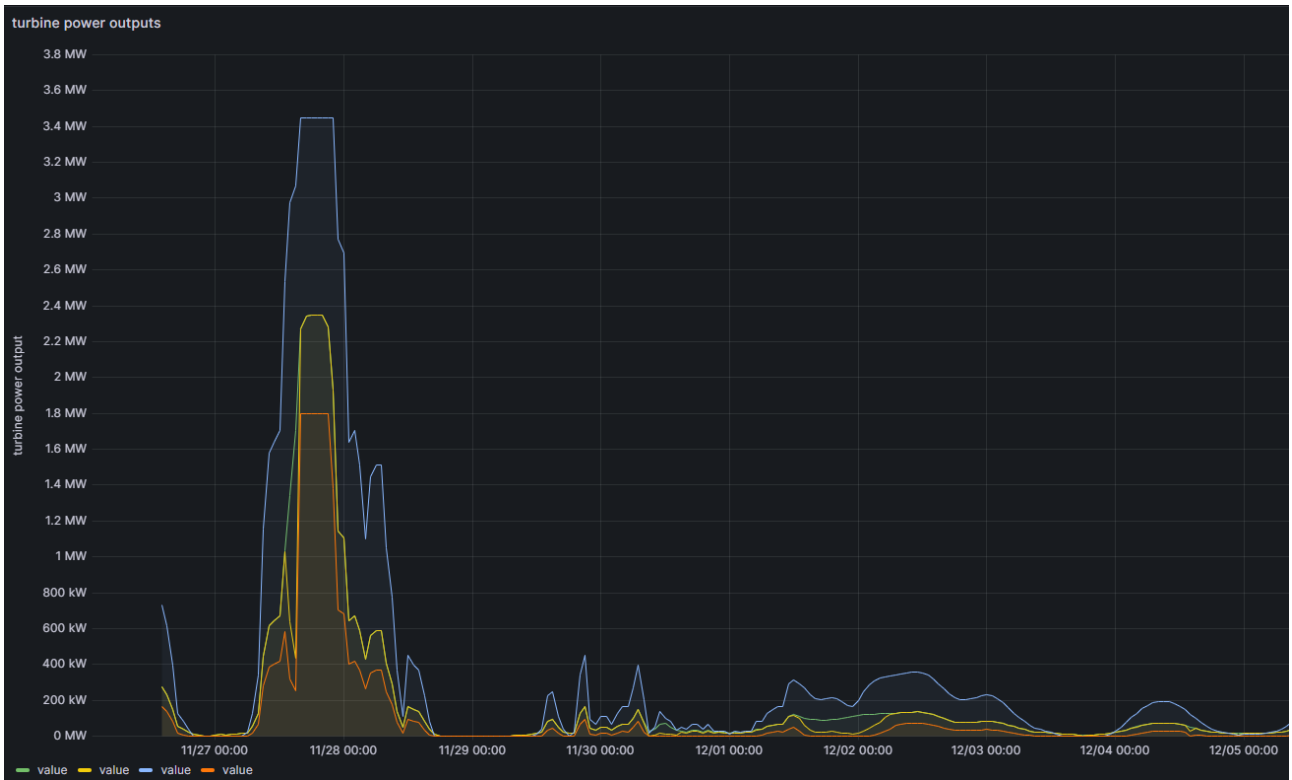


Figure 16: Typical forecast results from a wind park simulation as displayed by the dashboard.



Funded by  
the European Union

Project funded by

Schweizerische Eidgenossenschaft  
Confédération suisse  
Confederazione Svizzera  
Confederaziun svizra

Federal Department of Economic Affairs,  
Education and Research EAER  
State Secretariat for Education,  
Research and Innovation SERI

Swiss Confederation

## REFERENCES

---

- [1] M. Souppaya, K. Scarfone, and D. Dodson, "Secure Software Development Framework (SSDF) version 1.1 : recommendations for mitigating the risk of software vulnerabilities," National Institute of Standards and Technology (U.S.), Gaithersburg, MD, NIST SP 800-218, Feb. 2022. doi: 10.6028/NIST.SP.800-218.
- [2] D. Fucci, E. Alégroth, M. Felderer, and C. Johannesson, "Evaluating software security maturity using OWASP SAMM: Different approaches and stakeholders perceptions," *Journal of Systems and Software*, vol. 214, p. 112062, Aug. 2024, doi: 10.1016/j.jss.2024.112062.



**Funded by  
the European Union**

**Project funded by**



Schweizerische Eidgenossenschaft  
Confédération suisse  
Confederazione Svizzera  
Confederaziun svizra

Swiss Confederation

Federal Department of Economic Affairs,  
Education and Research EAER  
**State Secretariat for Education,  
Research and Innovation SERI**