Funded by the European Union

PROCTWIN

D8.2 Description of APIs, communication protocols, and interoperability

Version 1.0

Due date	31.12.2025
Dissemination level	Public
Work package title	Data management concepts
Lead editor	BFI
Internal reviewer	Swerim
Status	Final version

This project has received funding from the European Union under grant agreement NUMBER —101178721— ProcTwin

Authors

Name	Partner
Christoph Nölle	BFI
Petri Kannisto	BFI
Albert Fort	AQS

History

Date	Version	Authors	Description
27.10.2025	0.1	Christoph Nölle	Initial version
04.11.2025	0.2	Albert Fort	Updated Figure 4
04.12.2025	0.3	Petri Kannisto	Comments and smaller changes
09.12.2025	0.4	Christoph Nölle	Added Abstract, Outline, Conclusion, changes in the platform according to recent discussions.
11.12.2025	0.5	Albert Fort	Update Figure 4 and fill Sections 3.2, 3.3, 3.6 and update section 3.4
12.12.2025	0.6	Christoph Nölle	Added Section 4, accepted all changes.
15.12.2025	0.7	Olle Sandin, Frank Eriksson, Patrik Sidestam	Reviewed with a few comments and minor changes
15.12.2025	0.8	Christoph Nölle	Review changes accepted.
18.12.2025	1.0	Christoph Nölle	Final version

Abstract

This deliverable provides a specification of the ProcTwin data platform to be developed. It builds upon the earlier architecture description in deliverable D8.1. Whereas the architecture was meant to be abstract and focus on functionality, in this document the actual software components and communication protocols implementing these concepts are listed. The goal of the data platform is to provide access to process data from various sources, such as the Level 2 system, MES and ERP systems, and custom databases or files storing data generated by the new sensor systems installed during the project. Furthermore, it provides storage and visualisation capabilities for the simulation results and machine learning predictions developed.

PROCTWIN

Table of contents

Glossary

Abbreviation	Meaning
API	Application Programming Interface
ERP	Enterprise Resource Planning
ETL	Extract-Transform-Load
IAM	Identity and Access Management
JSON	JavaScript Object Notation
LDAP	Lightweight Directory Access Protocol
MES	Manufacturing Execution System
MQTT	Message Queueing Telemetry Transport
NFS	Network File System
ODD	OpenDataDiscovery
OT	Operational Technology
SDK	Software Development Kit
SQL	Structured Query Language

1. Introduction

1.1 Scope

This document aims to provide a specification of the component and platform design for the ProcTwin data platform. It defines the technologies to be used in the implementation phase of work package 9, including software components, application programming interfaces (API), data models, and communication protocols that enable interoperability between the parts of the overall system. Beyond that, the goal is also to define which software components to use concretely, with a preference towards open-source software.

Since software development is always iterative, this document is not meant to be static, but updates will likely be required during the project. Because there are no future deliverables for this purpose, the updates will be specified and communicated internally with tools, such as GitHub and SharePoint.

1.2 Deliverable role within the project

The present deliverable builds on the architecture specification of deliverable D8.1 and represents the final step in the requirements specification lifecycle for the ProcTwin data platform, consisting of the platform and components design. It serves as the main input to Task T9.1, the implementation of the data platform.

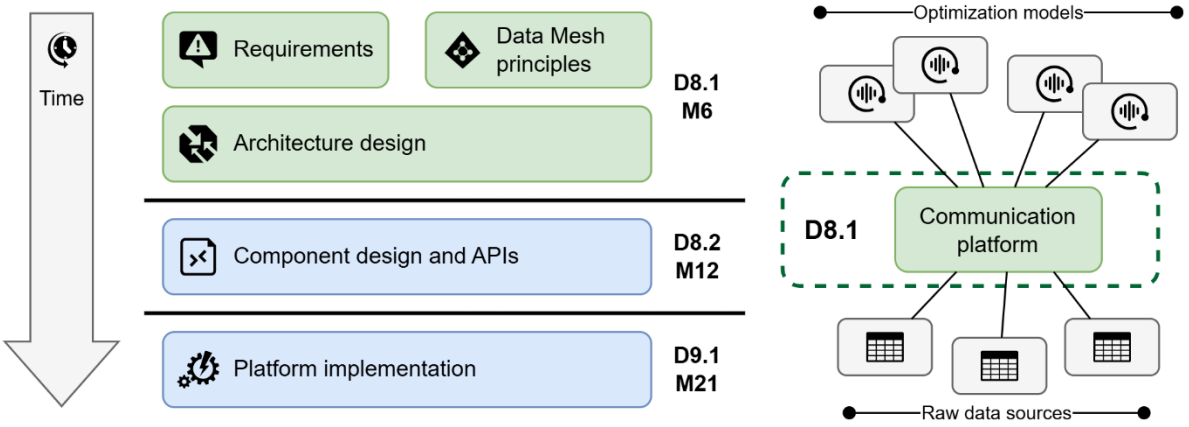

Figure 1. Role of the deliverables D8.1, D8.2 and D9.1 in the project. From ProcTwin D8.1: Description for software system architecture and security.

The specification thus paves the way for the physics-based simulations (WP6) and machine learning models (WP10-12) to access process data in a near real-time way, to store results of those models and from the newly installed sensors, and to calculate KPIs addressed by the project. Furthermore, the user interface to be specified and implemented in WP14 will consume data from the platform. Finally, the present specification represents an important prerequisite for the two project demonstrators to be developed in WP15.

1.3 Outline

The remainder of this document is structured as follows:

Section 2 recapitulates the architecture description of D8.1, introducing the different functional components required for the platform.

Section 3 describes the components and technologies that will be used for the platform implementation, as well as security measures foreseen.

2. Architecture overview

This section provides a brief recapitulation of the results from D8.1. An overview of the positioning of the data platform in the context of the data mesh is shown in Figure 2. The main goal of the platform is the collection and provision of data from various sources, such as process data from the automation system, and sensor data from new sensors installed during the project. The design builds upon the concept "Data Mesh", where the various data sources and data-oriented software components (such as optimization models) are portrayed as Data Products. These consume data to build additional value for either other components or the end user. The Data Mesh is primarily a governance concept for the overall principles, not limiting the system structure in any way. However, the Data Mesh expects a Self-serve Data Platform to help the Data Product developers with various tools. Related to this document, the "data platform" focuses on data integration and storage and is seen as a part of the Self-serve Data Platform. The parts not directly included in the data platform cover at least version control with git and any other structures supporting the development work.

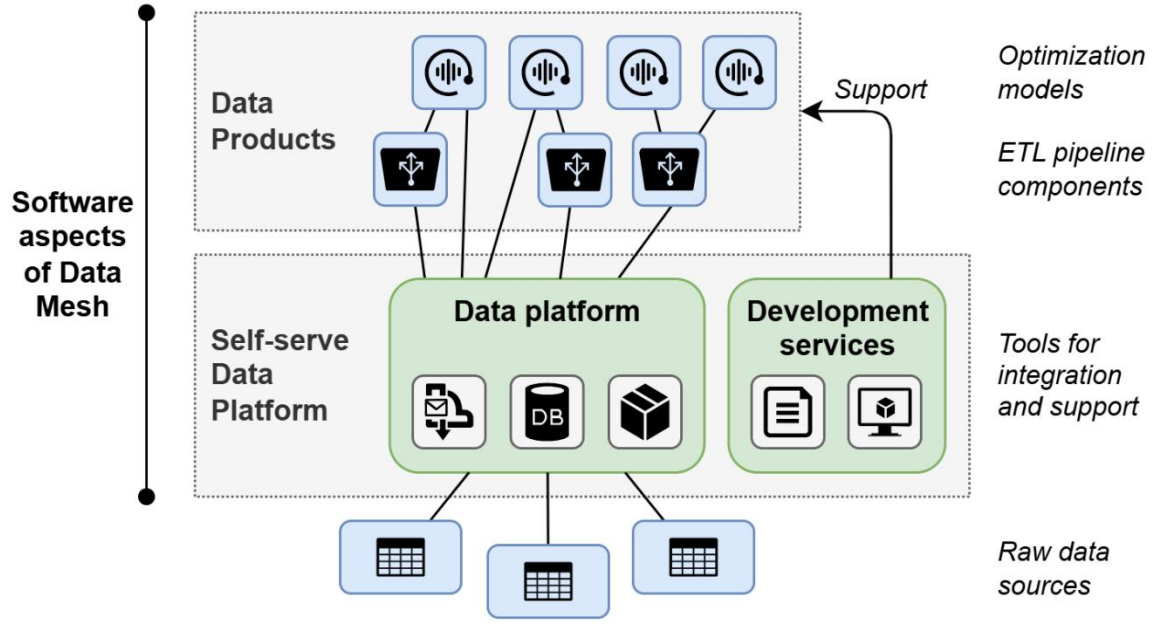

Figure 2. Software components in the data mesh. From ProcTwin D8.1: Description for software system architecture and security.

A view of the platform internals, as envisioned in D8.1, is shown in Figure 3. Besides the platform components itself, there are container orchestration and monitoring tools for operation.

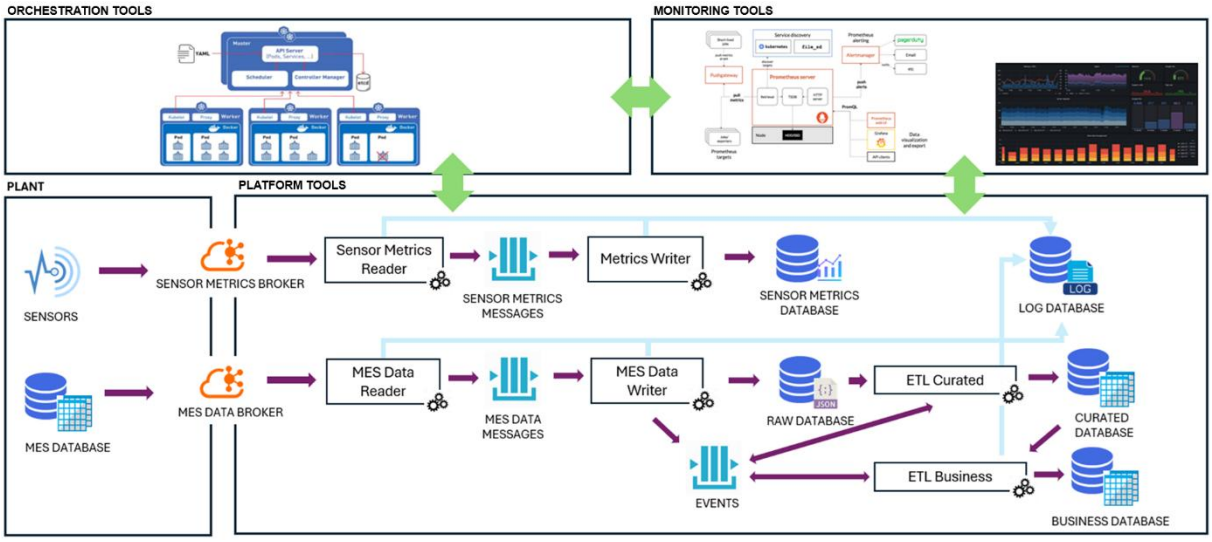

Figure 3. Target architecture of the ProcTwin data platform. From ProcTwin D8.1: Description for software system architecture and security.

The following components are foreseen:

- Data broker
  - Broker for sensor data
  - Broker for MES data / events
  - Broker for curated and business data
- Data storage
  - Sensor metrics database: a timeseries database.
  - Raw database: storage of incoming MES data / events.
  - Curated database: storing data in a structured, reusable way, according to a predefined schema.
  - Business database: stores business-relevant aggregated data, such as KPIs derived from the curated data.
  - Log database: for monitoring purposes.
- Extract-Transform-Load (ETL) pipeline:
  - Data readers and writers, responsible for transferring incoming data from the message broker to a message queue and finally to the persistence layer
  - ETL curated: transforms raw data to curated data
  - ETL business: transforms curated and raw data to business-relevant KPIs

The following component has been proposed, as well, in the document, although it does not appear in Figure 3:

- API
  - Dedicated interfaces for different types of data, such as time series vs. images, video, or structured data.
  - Data models specifying the structure of the data to be expected.

In addition, cross-cutting concerns must be addressed, such as IT security (in particular, authentication and authorization, encryption). This may require the introduction of additional components, such as an Identity and Access Management (IAM) tool or connectivity to an existing external tool, and additional means to protect the platform from malicious actors.

3. Platform specification

3.1 Main platform

The proposed implementation of the main platform is illustrated in Figure 4. It is based on the architecture overview of Figure 3 and adds the names of the components and communication protocols selected for the implementation of the different functionalities.

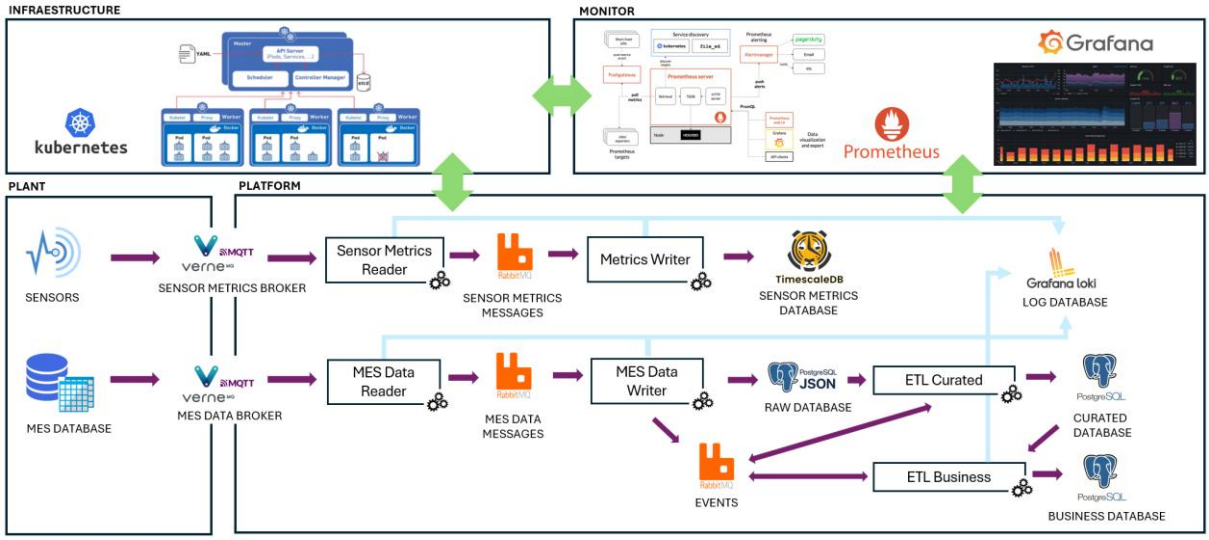

Figure 4. Software components for the ProcTwin data platform.

In the following subsections, the different parts of the platform are described.

3.2 Orchestration

For the deployment and execution of the data platform, we will use an open-source container orchestration system: Kubernetes.

Kubernetes automates the deployment, scaling, and management of containerized applications, allowing distributed applications to run efficiently while ensuring high availability, load balancing, fault recovery, and automatic scalability.

Kubernetes will be used to run the software components of the data platform, as well as simulation models and machine learning components developed during the project.

The choice of Kubernetes over other open-source products is based on:

- It offers better scalability and resilience compared to other products such as Docker Swarm or Nomad.
- It has the largest support community among orchestrators.
- It can be integrated with most cloud platforms (AWS, Azure, GCP, OpenShift).

The only drawback is that the initial configuration is complex, and it has a steeper learning curve compared to other similar products.

The design of the Kubernetes architecture to deploy the data platform would be as follows:

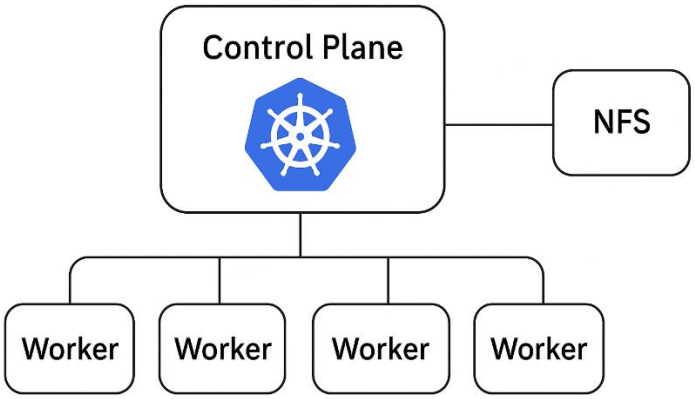

Figure 5. Kubernetes architecture design for the ProcTwin data platform.

The different servers required to deploy Kubernetes will be the following:

- **Control Plane:** Responsible for managing the entire Kubernetes infrastructure.
- **NFS:** The network file system, responsible for persisting all data from Kubernetes components (databases, queue systems, logs).
- **Workers (3–4 servers):** Responsible for running all components deployed on Kubernetes (the load balancing of these component instances is handled automatically by Kubernetes).

3.3 Observability

For monitoring the platform, we will rely on three main tools: **Prometheus**, **Grafana Loki**, and **Grafana**. Together, they form a complete observability stack that allows us to track metrics, collect logs, and visualize the overall status of the system.

Prometheus will be used to gather all metrics related to the health and performance of the different Kubernetes components and servers. It is an open-source monitoring and alerting system designed to collect real-time metrics using a pull model. Prometheus stores this information in a highly optimized time-series database, making it very efficient when performing queries. Thanks to its label-based architecture, it integrates seamlessly with Kubernetes, allowing metrics to be classified and filtered by service, pod, node, and many other attributes. Its widespread adoption in microservices and cloud-native environments has made it a cornerstone for modern observability.

For log management, we will use **Grafana Loki**, an open-source system developed by Grafana Labs and designed to work natively with both Prometheus and Grafana. Loki takes a more efficient approach to log indexing: instead of indexing the full text of each log entry, it only indexes labels, which significantly reduces storage requirements and operational costs. This makes Loki a highly scalable solution capable of supporting everything from small installations to large, distributed clusters. Because it integrates directly with Grafana, it enables unified queries of both logs and metrics, using LogQL, a query language inspired by PromQL.

Finally, **Grafana** will serve as the main interface for visualizing all this information. It is a powerful open-source platform for data analytics and dashboard creation, widely used in observability,

infrastructure monitoring, DevOps, IoT, and distributed systems. Grafana allows us to design interactive dashboards that combine graphs, tables, heatmaps, gauges, and many other visual components. It can connect to various data sources, including Prometheus, Loki, InfluxDB, PostgreSQL, and many others, making it extremely versatile. Thanks to its plugin ecosystem, Grafana can be easily extended with new visualization types and integrations. It is also highly flexible in terms of deployment, as it can run in a single container or form part of a more comprehensive observability stack on Kubernetes, Docker, Linux, Windows, or cloud environments.

Together, Prometheus, Loki, and Grafana provide a robust, scalable, and cost-effective solution for monitoring the platform, ensuring that we have full visibility into both system metrics and application logs.

3.4 Data ingestion

For data ingestion from the plant, the messaging protocol MQTT will be used, a standardized and widespread technology based on TCP/IP. It organizes messages into topics and provides a publish-subscribe mechanism. The data publishers themselves are outside the scope of the data platform proper and must be able to connect to the automation system as well as the data platform. The MQTT brokers will be realized by means of the open-source component Verne MQ.

After evaluating the various MQTT brokers, the decision was narrowed down to Mosquitto, EMQX, HiveMQ, and VerneMQ. Mosquitto was ruled out because, despite its popularity and compliance with open-source licensing, it does not provide the technical scalability required for large-scale deployments.

The remaining three brokers—EMQX, HiveMQ, and VerneMQ0—are designed to handle high message throughput and large numbers of concurrent connections. While all offer open-source versions, the fully-featured and technically advanced editions of EMQX and HiveMQ are only available in their commercial or enterprise versions.

In contrast, VerneMQ provides a complete feature set under a fully open-source license, making it the preferred choice for high-scale, open-source deployments.

There are two qualitatively different input streams that need to be handled by the MQTT brokers, an event-based stream with document-shaped payloads that, for instance, informs about new heat data that is provided at the continuous caster or a new plate ready for quenching, and a sensor stream with periodically arriving timeseries data. Besides having different characteristics in terms of data volume and update frequency, it is very important that no data from the event stream is lost, whereas a lost package of sensor data is less critical. It was therefore decided to use two different data brokers as endpoints for those streams, with different configurations.

Data ingested into the broker is then transferred to a persistent message queue by means of a data reader component. The message queues will be realized in terms of the RabbitMQ open-source component. Finally, messages will be transferred from the queues to the persistence layer by two dedicated message writers.

3.5 Persistence

The different databases will be implemented as follows. The open-source database PostgreSQL provides the curated and business databases, both of which are based on a predefined schema.

For the raw database, the use of a document database such as MongoDB has been proposed, which is purpose-built for storing data without a predefined schema. There are two drawbacks of this approach, however. On the one hand, MongoDB as the most mature document database, in its current version is published under a source-available license with strict copyleft requirements, which would permit the

anticipated use in the project. Nevertheless, it restricts the possible uses of the open version of the database and would practically not permit the forking of the software in case the open version is discontinued. On the other hand, the introduction of another type of database server implies additional maintenance overhead compared to the use of another instance of PostgreSQL. Although the latter was traditionally not considered suitable for storing schema-less data, it does allow for JSON fields, i.e., strings representing documents with arbitrary keys, and in more recent versions even allows indexing those keys, applying filters in queries, and provides efficient storage. While the technological gap between a dedicated document-oriented database like MongoDB and the SQL store PostgreSQL for document storage is certainly still present, the advantage of reusing technology and true open-source software in combination with improved JSON storage have led to the decision for PostgreSQL here.

Regarding the timeseries database, three options have been evaluated. InfluxDB has traditionally been considered the go-to open-source timeseries database, and is still today in wide use. With the advent of its third major version, the split between open-source and enterprise features has been significantly altered, however, and it is unclear whether the open-source version still satisfies the requirements of a production-grade database[1]. Timescale is another well-known timeseries database, implemented as an extension to PostgreSQL for efficient storage and handling of timeseries data. Besides the hosted edition, it is made available in a Community edition under a source-available license, restricting potential service offerings of Timescale as a service, compared to a true open-source license. The third timeseries database considered is QuestDB, which is available under a liberal open-source license (Apache v2). It reuses the PostgreSQL wire protocol. Due to the compatibility with the the curated and business databases, it was decided to use the Timescale database for the data platform.

3.6 ETL

Raw data is transformed into curated (technical) data and business relevant data by means of two ETL pipelines, with results stored in two PostgreSQL databases.

The ETL pipelines will have the primary function of transforming and/or validating data formats according to how the data will be used.

The ETL Curated pipeline will be responsible for refining and validating the data received in its RAW form, and for recording all these values in the "Curated" database. This ensures that, without altering the structure of the data received, no incorrect or invalidly formatted values are stored in the database.

The ETL Business pipeline will handle transforming and adapting the data stored in the Curated database into the model and structure defined in the Business (Final) database.

Both pipelines will be implemented in a generic and configurable way, designed to accommodate any type of data or data model. In cases where very specific data transformation or aggregation with sensor data is required, the platform will allow these functionalities to be extended by adding new ETL pipelines with a more customized implementation tailored to the specific use case.

3.7 Communication Protocols

Regarding data retrieval from the platform, it was decided to directly access the internal databases via their respective wire protocols, i.e., the PostgreSQL protocol in our case, instead of introducing a new component that provides application programming interfaces (APIs) to external software and translates API calls to database queries. While this has the drawback of strict coupling of applications

[1] Initially, the query API of the open-source version had been limited to returning the last 72 hours. While this limitation has been relaxed in the meantime, the compaction algorithm that enables efficient handling of queries for larger periods of time is only part of the enterprise edition, leaving the open-source edition as more of a test version.

to the database schema, which is considered a bad practice for production-grade software, it allows for fast iterations in the schema development and reduces the number of components that need to be operated, which is a clear plus in a research project.

Instead of providing an API component in the platform, it is foreseen to support the application developers through the introduction of a software development kit (SDK).

3.8 Security

Several steps will be taken to ensure the security of the data platform.

- **Network segmentation and firewall rules** will be used to ensure that while data from the automation system can be transferred to the data platform, connectivity in the reverse direction is blocked, protecting the production-critical operational technology (OT). Furthermore, the platforms may be installed on-premises at the involved plants and inaccessible from the public internet.
- **Kubernetes network policies** ensure that services are only accessible to other services they need to communicate with.
- **Authentication and authorization:** User accessible services, such as the databases of the data platform and the dashboarding tool, are configured to enforce user authentication and access checks. Where possible, an integration with an existing IAM system will be preferred. For instance, PostgreSQL supports authentication via the LDAP or OAuth 2.0 protocols, which are provided by many enterprise IAMs. In other cases, dedicated database or service users will be created and fine-grained access control rules implemented. Authorization will also be enforced for service-to-service communication, leveraging a zero-trust architecture for the data platform.
- **Software supply chain security** is taken into account by continuously checking the used open-source dependencies for known vulnerabilities and ensuring that updates are quickly adopted. Automated scanning of the containers for known vulnerabilities will be employed before deployment to the productive Kubernetes cluster.
- **Web security:** for the implementation of the frontend in WP14, best practices of web development will be followed, to avoid common pitfalls [2], such as cross-site scripting vulnerabilities.
- **Container security:** The OWASP guidelines for Docker[3] and Kubernetes[4] security provide best practices on how to avoid container vulnerabilities and operate containerized software in a secure way.

Besides these general principles regarding IT security, deployment of the platform in the company networks operating the demo plants requires conformance to their respective IT security standards.

3.9 Catalog and data products

Since the project aims to showcase a data mesh methodology, integration of a data product catalog and the provision of data products by different domain teams is one goal. These data products encompass among others specific datasets, which typically live in the curated or business databases, but they can also include offline, file-based datasets, software components, and other artifacts.

[2] https://owasp.org/Top10/
[3] https://cheatsheetseries.owasp.org/cheatsheets/Docker_Security_Cheat_Sheet.html
[4] https://cheatsheetseries.owasp.org/cheatsheets/Kubernetes_Security_Cheat_Sheet.html

The following is a selected set of open-source software for implementing a data catalog and related features:

- **OpenMetadata**[5]: a platform for data discovery, observability, and governance. It enables in particular the creation of data products[6].
- **OpenDataDiscovery (ODD) Platform**[7]: ODD describes the process of gathering metadata from data storages/sources such as data lakes and data warehouses, data discovery processes through push and pull models and APIs that should be provided. Metadata is stored in a data catalogue.
- **DataHub**[8]: a metadata platform for data & AI

Additional software and an overview of supported features can be found in the Github repository https://github.com/opendatadiscovery/awesome-data-catalogs, by the maintainers of the ODD Platform.

Within WP9, the use of these tools as data catalogs for the data platform will be trialled.

[5] https://open-metadata.org/
[6] https://docs.open-metadata.org/latest/how-to-guides/data-governance/domains-&-data-products/data-products
[7] https://opendatadiscovery.org/
[8] https://datahub.com/

4. Usage in the project

The data platform is used for storing both process data and new sensor data, as well as model results. Sensor data ingestion is part of the sensor metric ingestion route in Figure 4. The interaction of physics-based simulation models and data-driven models with the platform is illustrated in Figure 6. Both types of applications will preferably run in the container orchestration platform, like the platform components themselves. They access data mostly from the sensor metrics database and the curated database, whereas their results are directly fed to the raw database. Dedicated ETL components then check the new results and write them back to the curated database, and potentially also some KPIs directly to the business database (not shown).

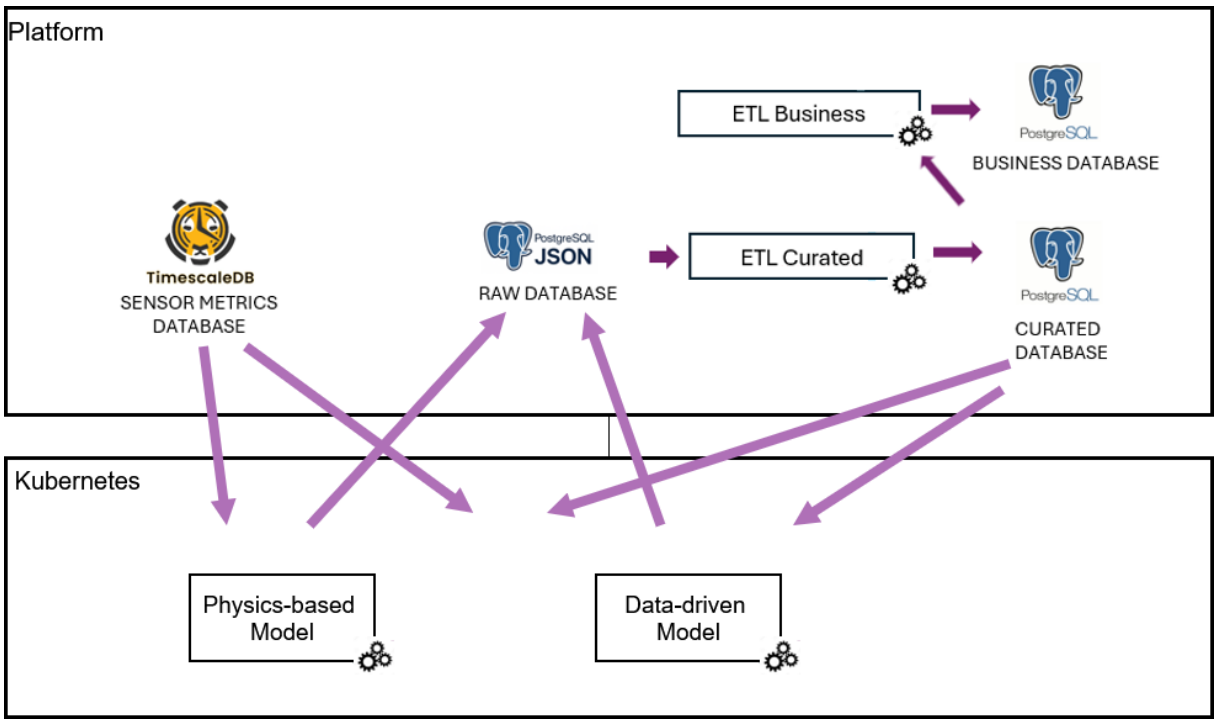

Figure 6. Interaction of applications with the platform storage layer.

The results of the physics-based models in the curated database can then be fed into the data-driven models, serving as soft sensor input.

5. Conclusion

The present document specifies the technology to be used for the implementation of the ProcTwin data platform. It is based on a solid architectural design developed previously in D8.1 and represents a modern software platform consisting almost exclusively of open-source components. It is foreseen to publish the Kubernetes configuration for the data platform on a platform such as Github, so that it can be reused in other environments.

The platform design considers the different priorities of expected input streams and comes with comprehensive monitoring capabilities. It is therefore meant to provide a resilient, production-grade platform for the project demonstrators, with the possibility to remain in operation after the end of the project.

The integration of the platform with data mesh technology and governance rules will be further refined during the project, in particular the use of a data product catalogue for documentation and discoverability of data products.