RE4DY

MANUFACTURING DATA NETWORKS

RE4DY TOOLKIT

Name of the Tool	Data Container
Tool Owner	Industry Commons Foundation
Version	1.0
Date	Nov 2025
Version	V1.0

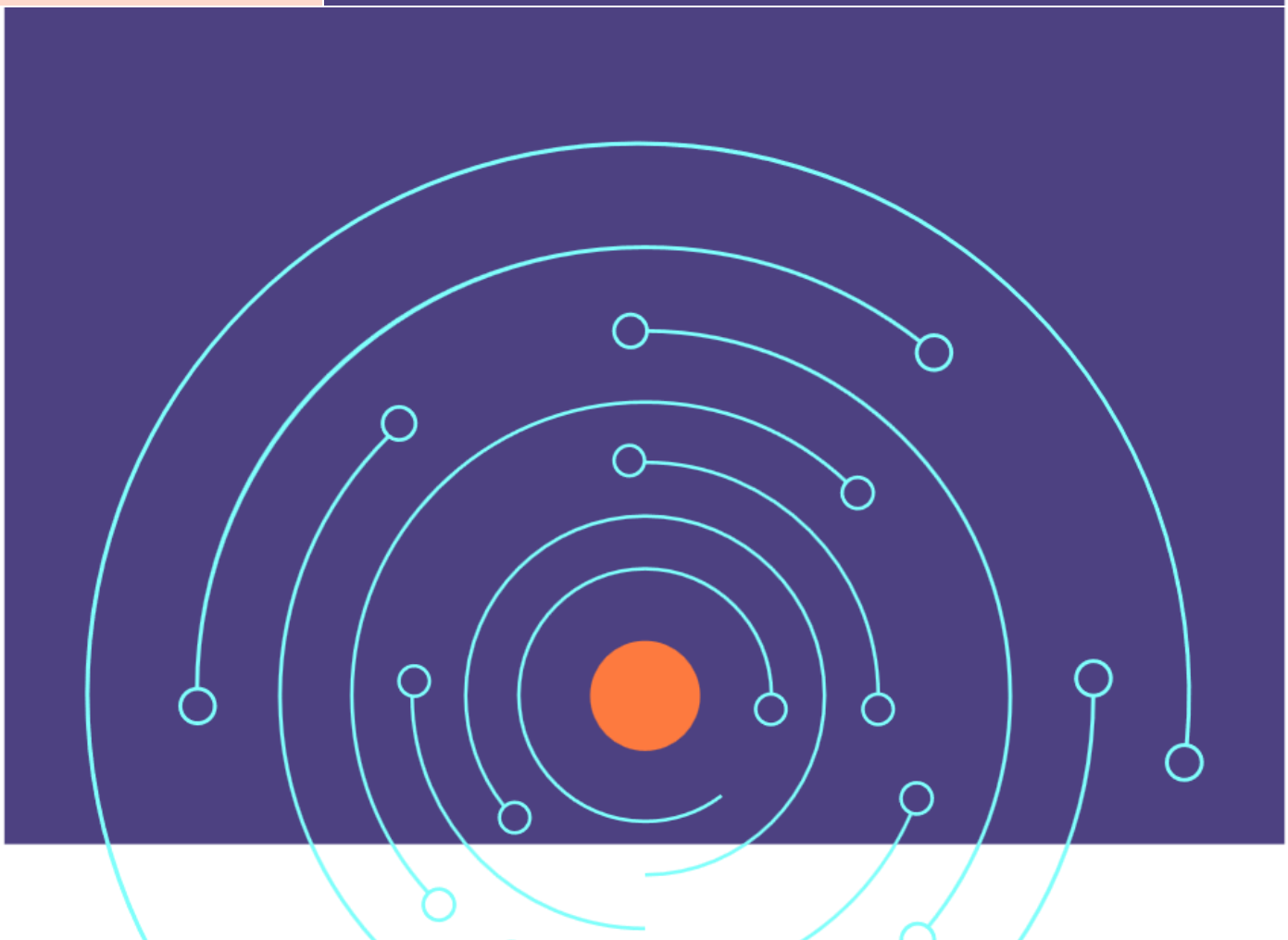

Table of contents

1. Component Description

The Data Containers (DC) provide access to the data following the Data as a Product approach while, at the same time, creating an abstraction layer that will hide all the underlying technical complexity from the users. DC will allow the applications to access the data in a trusted and reliable way regardless of the location (Edge or Cloud) and particularities of the data sources. The DC will provide the data according to the consumer's requirements in terms of format and data quality, with the proper considerations about security, privacy, and performance.

2. Input

The DC should be able to obtain the data from the sources (directly or indirectly) and call the necessary components of the RE4DY architecture in order to serve the data following the Data as a Product approach.

Inputs of the DC API:

- **Dataset id**
- **Format**: defines the output data format (for example, JSON, Parquet or CSV).
- **Rules**: allow the data consumer to select a subset of the available data. Rules can also be used to remove outliers from the data. A rule consists of three elements:
    - **Subject:** variable where the rule is applied.
    - **Operator:** the allowed operators are the following: "<=", ">=", "<", ">", "==", "!=", "or", "++" (include only the columns whose names are specified in the "object" value), and "—"(do not include the columns whose names are specified in the "object" value).
    - **Object:** value.

3. Output

The Data Containers will provide data and metadata.

Metadata request

The metadata will be sent in the HTTP response in JSON format.

Data request

- API response: indicates whether the request was completed successfully or not.
- Notification: is sent to the specified HTTP endpoint once the process is completed and data is available (in the prototype, the URL of this endpoint is defined in the DC configuration). Example:

```
{
```

```
    "message": "Data sent successfully to FTP server",
    "success": "true"
}
```

- Data: The requested data will be served following the consumer's requirements in terms of format and data quality. The current version makes the data available for the consumer using an FTP server.

4. Information Flow

Obtaining cleansed and preprocessed historical data from storage

The flow shown in Figure 1 describes a particular scenario in which a cleansed and preprocessed dataset is available in the data storage. In that case, the DC can retrieve the data from the storage, apply the requested filtering rules, and format and serve it to the client. This scenario has been implemented in the prototype. To facilitate the management of big datasets, the data is sent to a repository (FTP or SFTP server) and the client receives a notification when the data is available.

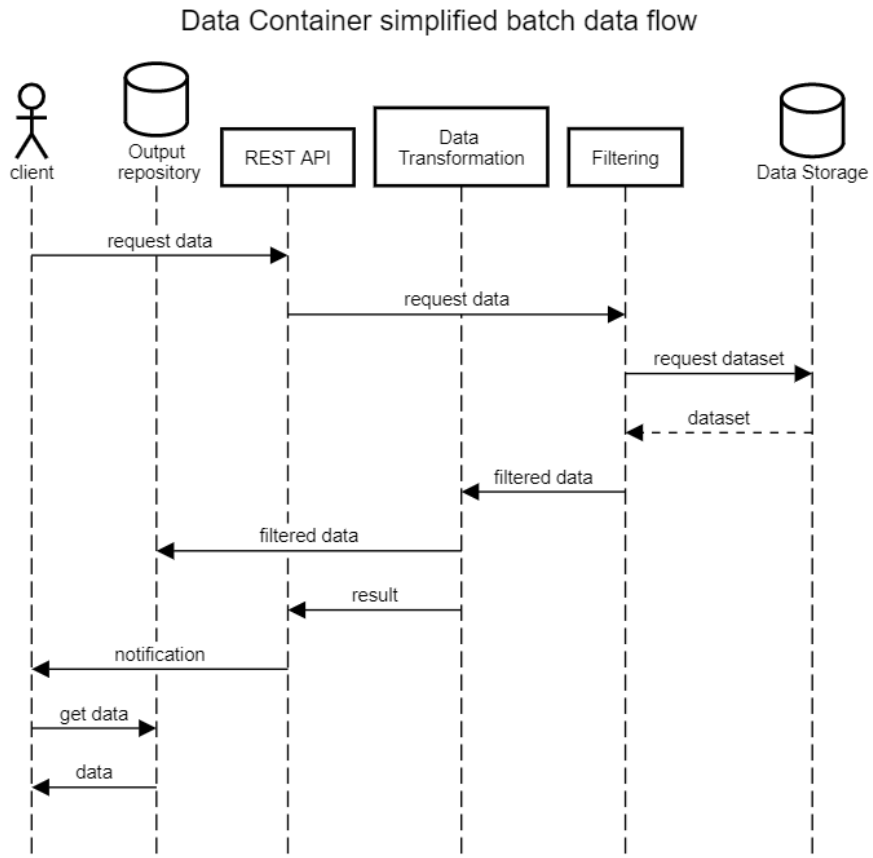

Figure 1 UML diagram of the process of retrieving data from the storage and serving it to the client

Obtaining historical data from the source

The flow shown in Figure 2 illustrates the overall process. Hence, it includes calls to other blocks of the RE4DY architecture, such as Data Ingestion and Semantic Transformation. To facilitate the management of big datasets, the data can be sent to a repository (such as an FTP or SFTP server). In that case, the DC sends a notification to the client when the data is available.

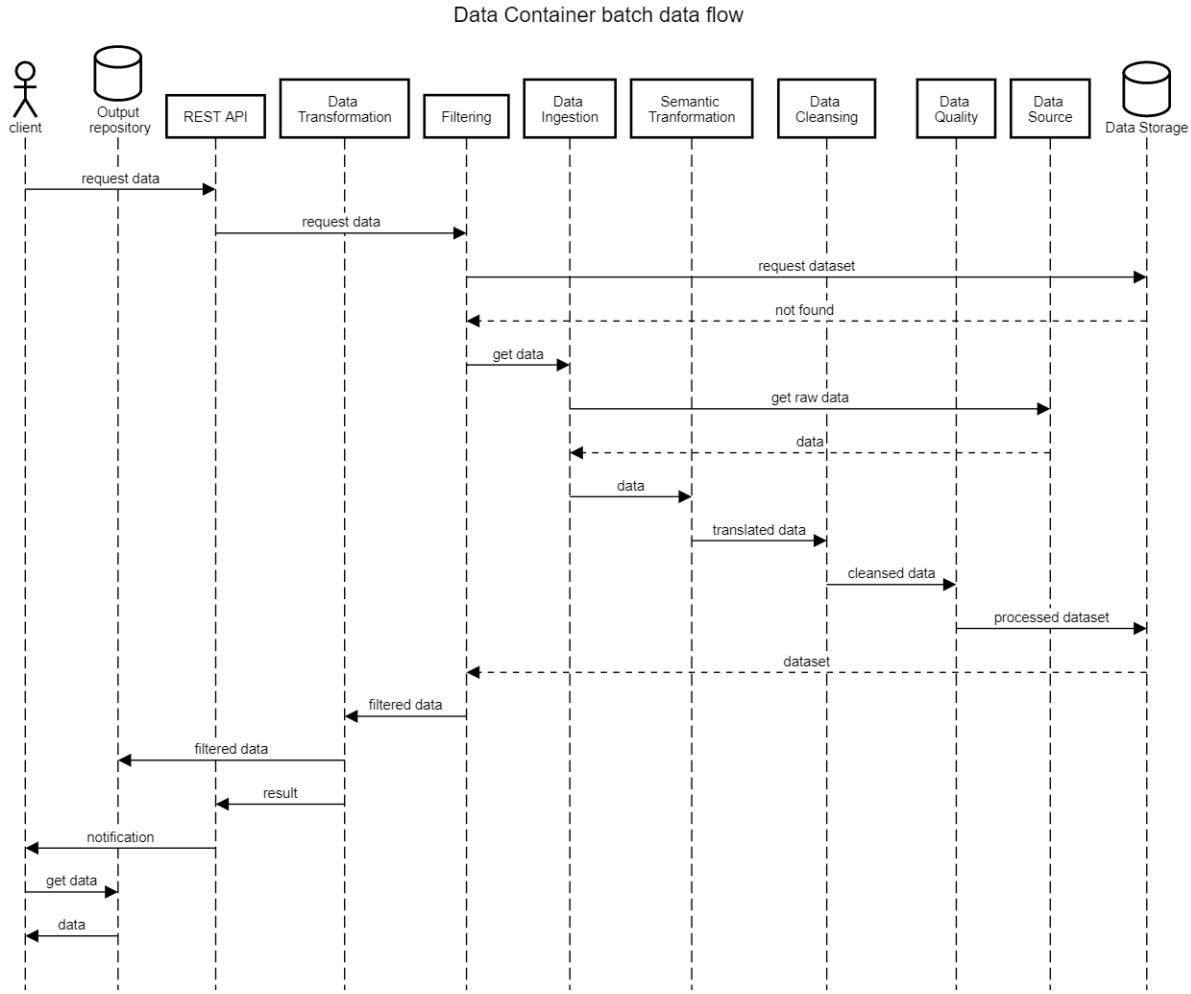

Figure 2 UML diagram of the complete process of retrieving and serving batch data in a generic Data Container.

5. Internal Architecture

Data Containers expose a REST API to enable access to the data and metadata. The control module manages the DC configuration and ensures that the conditions defined in the DCP are applied. When a request is received, the Data Processing module performs the necessary calls to other components of the RE4DY architecture in order to retrieve the requested data and ensure that the client's requirements are met. This module can also apply filtering rules to select only a subset of the available data. Finally, the Data Transformation module prepares the data and converts it into the format requested by the client.

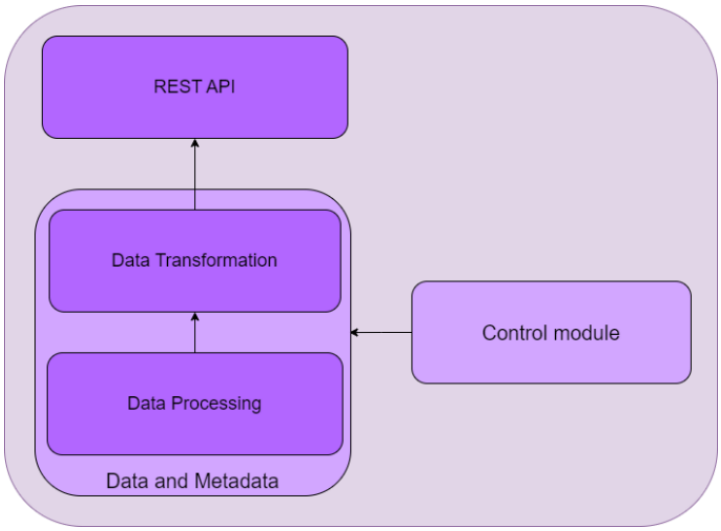

Figure 3 Internal architecture of a generic Data Container

6. API

- `GET /metadata`
  - **Description**
    This endpoint provides metadata associated to the datasets provided by the DC. The input parameter “dataset_id” identifies the dataset for which the metadata is requested.
  - **Input**
    - dataset_id
  - **Output**
    {

    "variables": 11,

    "rows": 1,

    "size": 1291,

    "lastUpdate": "2023-06-05T15:31:16Z",

    "variablesNames": ["var1", “var2", …]

    }

- `POST /vdc`
  - **Description**
    This endpoint is used by the data consumer to request data from the DC. The input parameter “dataset_id” identifies the dataset. The parameter “format” defines the selected output format. The JSON array “rules” defines the filtering rules that will be applied to the selected dataset. A notification will be sent to the URL specified in the parameter “notification_url” when the dataset is available in the output repository (FTP server).

- **Input**


```
{
 "dataset_id": "FaultySteelPlates",
 "rules": [
    {
    "name": "test-rule",
    "rule": {
      "subject_column": "TypeOfSteel_A300",
      "operator": "==",
      "object": 1
    }
  }
 ],
 "format": "json",
 "notification_url": "http://example.com/notify "
}
```


- **Output**


```
{
 "result rows": 777,
 "message": "Request received successfully. Preparing results...",
 "success": true
}
```


7. Implementation Technology

Through the Data Containers, it should be possible to define flows that connect different elements of the RE4DY architecture to enable access to the data following the DaaP principles. Thus, the preferred implementation for the DCs is based on flow-oriented programming frameworks such as Apache NiFi or Node-RED. The current implementation has been created using NiFi and is deployed on Docker.

8. Comments

The current implementation is a prototype. Specific Data Containers will be developed for each use case.

At this point, only the flows for historical data have been defined. The use of DC with near real-time data will be defined in future versions.