# D3.8 – i4Q DATA REPOSITORY

WP3 – BUILD: Manufacturing Data Quality

i4Q has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 958205.

## Document Information

| | | | |
|---|---|---|---|
| GRANT AGREEMENT NUMBER | 958205 | ACRONYM | i4Q |
| FULL TITLE | Industrial Data Services for Quality Control in Smart Manufacturing | | |
| START DATE | 01-01-2021 | DURATION | 36 months |
| PROJECT URL | https://www.i4q-project.eu/ | | |
| DELIVERABLE | D3.8 i4Q Data Repository | | |
| WORK PACKAGE | WP3 – BUILD: Manufacturing Data Quality | | |
| DATE OF DELIVERY | CONTRACTUAL: June 2022 | ACTUAL: June 2022 | |
| NATURE | Other | DISSEMINATION LEVEL | Public |
| LEAD BENEFICIARY | ITI | | |
| RESPONSIBLE AUTHOR | Jordi Arjona (ITI), Santiago Gálvez (ITI), Emili Miedes (ITI), Sonia Santiago (ITI) | | |
| CONTRIBUTIONS FROM | 2- ENG, 5-KBZ, 11-UNI, 12-TIAG, 22-RIAS, | | |
| TARGET AUDIENCE | 1) i4Q Project partners; 2) industrial community; 3) other H2020 funded projects; 4) scientific community | | |
| DELIVERABLE CONTEXT/ DEPENDENCIES | This deliverable is a public document developed as part of "*Task 3.5 - Manufacturing Data Storage and Use*", that presents a technical overview of the i4Q Data Repository solution (i4Q$^{DR}$). This deliverable has no preceding documents but will have a new iteration called D3.16 "i4Q Data Repository v2", that will be delivered at M24. D3.8 receives input from deliverables D1.9, D2.6, D2.7, and D3.7. Moreover, it provides feedback to D3.7. | | |
| EXTERNAL ANNEXES/ SUPPORTING DOCUMENTS | None | | |
| READING NOTES | None | | |
| ABSTRACT | This deliverable presents a technical overview of the i4Q Data Repository solution (i4Q$^{DR}$), including an explanation of its mapping against the i4Q Reference Architecture. Furthermore, it provides specific information regarding its implementation status up to M18, describing developments performed so far. In this regard, this document also explains the remaining implementation work, that will be addressed in the rest of task. | | |

## Document History

| VERSION | ISSUE DATE | STAGE | DESCRIPTION | CONTRIBUTOR |
|---|---|---|---|---|
| 0.1 | 05-May-2022 | ToC | First Version of Table of Contents | ITI |
| 0.2 | 11-May-2022 | 1st Draft | Preliminary content for all sections | ITI |
| 0.3 | 13-May-2022 | 2nd Draft | Version after task partners contributions | ITI, ENG, UNI, KBZ |
| 0.4 | 16-May-2022 | 3rd Draft | Version ready for internal review | ITI |
| 0.5 | 26-May-2022 | Internal review | Review and comments | TIAS, RIAS, |
| 0.6 | 12-Jun-2022 | 4th Draft | Updated version based on reviews | ITI |
| 1.0 | 30-Jun-2022 | Final Draft | Final quality check and issue of final document | CERTH |

## Disclaimer



## Copyright message

# TABLE OF CONTENTS



# LIST OF FIGURES



# LIST OF TABLES

# ABBREVIATIONS/ACRONYMS

| | |
|---|---|
| **API** | Application Programming Interface |
| **DBMS** | Data Base Management System |
| **DIT** | Data Integration and Transformation |
| **DR** | Data Repository |
| **DSS** | Decision Support System |
| **HA** | High Availability |
| **HA+Sec** | High Availability with Security |
| **HTTP** | Hypertext Transfer Protocol |
| **JSON** | JavaScript Object Notation |
| **PDF** | Portable Document Format |
| **REST** | Representational State Transfer |
| **REST API** | RESTful Application Programming Interface |
| **SQL** | Structured Query Language |
| **SS** | Single Server |
| **SS+Sec** | Single Server with Security |
| **TLS** | Transport Layer Security |

# Executive summary

**D3.8** delivers the first release of the i4Q Data Repository Solution and presents an executive explanation of the **i4Q Data Repository** (i4Q$^{DR}$) solution. More specifically, it provides a general description of the solution, its technical specifications, a report on its implementation status, and an explanation on the next planned steps. The source code of the i4Q$^{DR}$ Solution is available in a private repository of Gitlab at: https://gitlab.com/i4q/dr.

This document is complemented by the technical documentation associated to the i4Q$^{DR}$ Solution, which is deployed on the website http://i4q.upv.es. This website contains the information of all the i4Q Solutions developed in the project "Industrial Data Services for Quality Control in Smart Manufacturing" (i4Q). The i4Q$^{DR}$ Solution's technical documentation is publicly available at http://i4q.upv.es/8_i4Q_DR/index.html and provides information regarding the topics listed below:

- General description
- Features
- Images
- Authors
- Licensing
- Pricing
- Installation requirements
- Installation Instructions
- Technical specifications of the solution
- User manual

## Document structure

**Section 1:** Contains a general description of the i4Q **Data Repository**, providing an overview and the list of features. It is addressed to final users of the i4Q Solution.

**Section 2:** Contains the technical specifications of the i4Q **Data Repository**, providing an overview and its architecture diagram. It is mainly addressed to software developers.

**Section 3:** Details the implementation of the i4Q **Data Repository**, explaining the current status, next steps and summarizing the implementation history. Similarly, as Section 2, the content of this section is mainly addressed to software developers.

**Section 4:** Provides the conclusions.

**APPENDIX I:** Provides the PDF version of the i4Q **Data Repository** technical web documentation, which can be accessed online at: http://i4q.upv.es/8_i4Q_DR/index.html

# 1. General Description

## 1.1 Overview

The **i4Q Data Repository** ($i4Q^{DR}$) is a distributed storage system that will oversee receiving, storing, and serving the data in an appropriate way to other solutions. Note that these operations will be performed according to standard data storage system's mechanisms so that no specific data transformations will be applied. Indeed, there is another solution for this purpose, namely the i4Q Data Integration and Transformation Services solution ($i4Q^{DIT}$) which is presented in deliverable D4.1 [1]. The $i4Q^{DR}$ solution is suitable to support and enhance a high degree of digitization in companies with most manufacturing devices acting as sensors or actuators and generating vast amounts of data.

Firstly, $i4Q^{DR}$ is expected to absorb large volumes of data coming into the system at high speeds[1]. However, the demand of computing resources will vary over the time. Thus, $i4Q^{DR}$ is expected to adapt its computing resources to the actual demand at a given moment, so that it can use additional resources, when necessary. These resources could either be local to the factory or remote, such as public or private clouds depending on required operational and functional latencies.

Furthermore, $i4Q^{DR}$ provides the proper tools for administrators to characterize and transform the information contained inside it. In this regard, the $i4Q^{DR}$ allows exporting data in a different format to the one in which it was stored. Moreover, since $i4Q^{DR}$ will enable access to different data sources, data from a given data source can be enriched by correlating it with data from another data source. This solution is also useful for data scientists, who can use data stored in $i4Q^{DR}$ in their experimentations.

Finally, this solution includes some features to enhance secure data management. First, it will oversee data protection, serving as a secure data vault system for the information. This can be achieved by encrypting data, both in flight and at rest. Furthermore, $i4Q^{DR}$ will be in charge of ensuring administration of regulated access to the data and the related tools, so that only allowed entities are able to do so.

[1] More specific details on this aspects will be provided in the next version of this deliverable, D3.16,

## 1.2 Features

i4Q[DR] will include the features explained below:

- An *access control mechanism*, to ensure that only authorised entities have access to the data and the related tools. i4Q[DR] administrator users with the appropriate permissions will be able to configure this access control, in order to grant access to the allowed entities.
- Tools and technologies to *manage structured data*. This feature will be offered by means of DBMSs that may be relational SQL-based, document (e.g., JSON-based), general NoSQL tools, etc.
- Tools and technologies to *manage blobs*. This feature will be offered through tools that offer support for blobs like some general-purpose DBMSs or even specific ones (e.g., Minio).
- Mechanisms to *query* the stored data and retrieve the results of such queries, for both structured data and blobs, in an efficient way.
- Mechanisms to *import/export data* to/from i4Q[DR] to ease the interoperability of this solution with others.
- Mechanisms to manage the data repository itself.

# 2. Technical Specifications

## 2.1 Overview

The **i4Q Data Repository** (i4Q$^{DR}$) is aimed at transversally providing, in a centralized fashion, the functionality related to the storage of data in the whole i4Q system. Indeed, i4Q$^{DR}$ is involved in all pilots and is expected to interoperate with a large subset of the i4Q solutions.

The central nature of this solution allows taking and applying in a centralized manner some decisions related to the organization, management, and access to the data. For instance, access control criteria and policies criteria related to high availability and fault tolerance of the data storage tools, etc. These decisions are required for the proper design and the implementation of certain mechanisms related to data storage. Therefore, the central nature of this solution reduces the complexity of such decision-making processes.

Moreover, when it comes to putting into practice mechanisms and tools, this central nature of the solution avoids the duplication of efforts and more importantly, some problems derived from such duplicity, starting from having divergent criteria and ending in devoting duplicated efforts to the same tasks.

## 2.2 Architecture Diagram

The i4Q$^{DR}$ solution is mapped against different sub-components of the i4Q Architectural Framework presented in D2.7 [2], which are summarized in an illustrative way in Figure 1. First of all, since the i4Q$^{DR}$ solutions covers the data storage requirements of the i4Q system, it is mapped to the *"Data Brokering and Storage"* sub-component of the **Platform Tier**. This mapping is highlighted in green colour in Figure 1.

Secondly, considering the functionalities which the i4Q$^{DR}$ relies on, we can define mappings to some sub-components of the **Edge Tier**, namely, the *"Distributed Computing"*, the *"Data Collection"* and the *"Data Management"*. More specifically:

- The mapping to the *"Distributed Computing"*, sub-component provides the i4Q$^{DR}$ the necessary execution environments (containers, virtual machines, etc.) to support data replication.
- The mapping to the *"Data Collection"* sub-component supplies the flows of data to store.
- The mapping to the *"Data Management"* sub-component enables the definition of the structures of the data to store and the queries to execute to retrieve data.

These three mappings are highlighted in orange colour in Figure 1.

Finally, taking into account that the i4Q$^{DR}$ provides data storage services to any other solution requiring them, we can define mappings to any sub-component of the **Enterprise** and **Edge Tiers**. In Figure 1 we have highlighted these mappings in red colour.

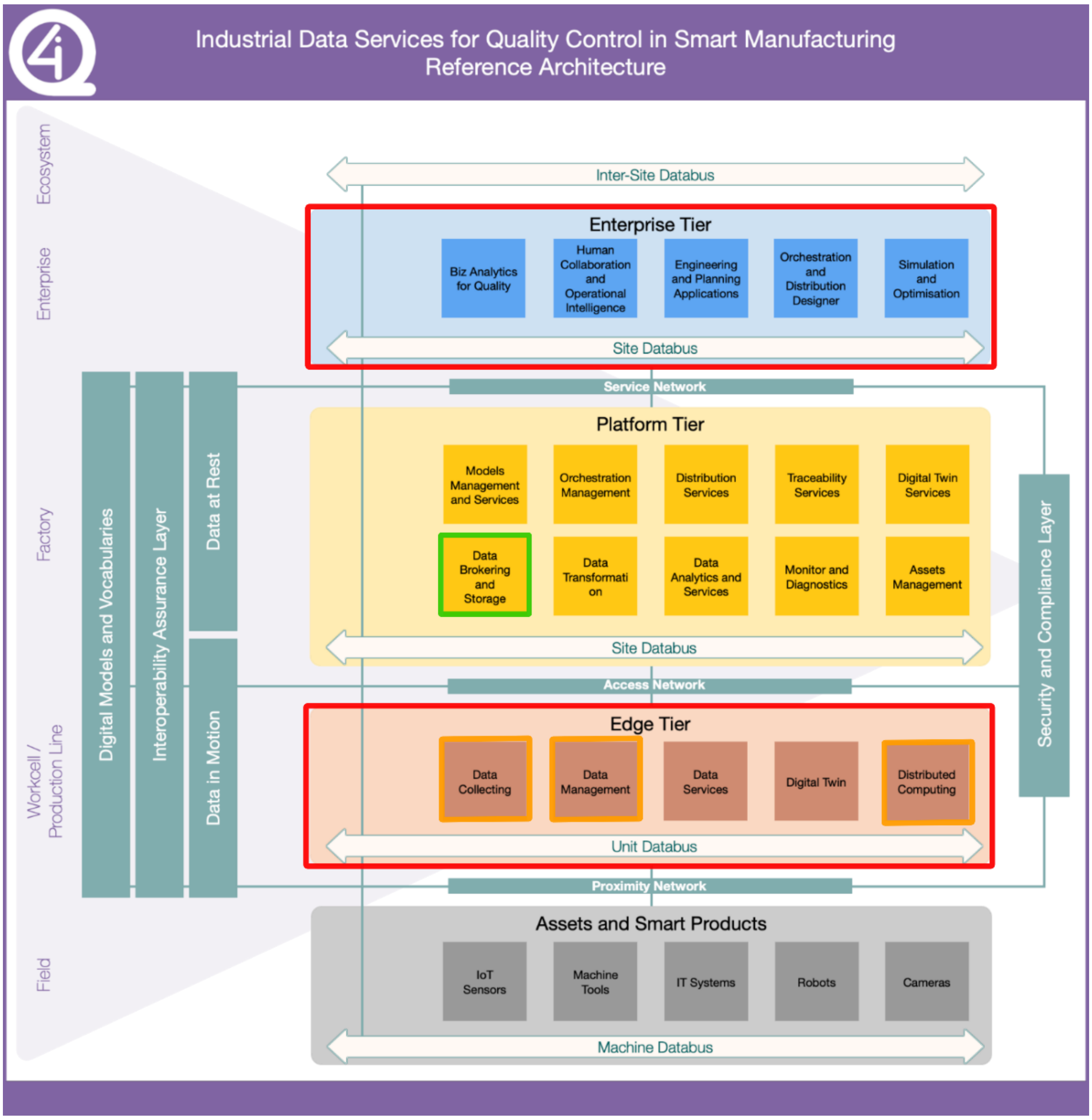



**Figure 1**. Mappings of i4Q[DR] against the i4Q Reference Architecture

# 3. Implementation Status

This section provides detailed information on the implementation of the i4Q$^{DR}$ solution. Note, however, that this is a work in progress, since task T3.5 will finish at M24. That is, the full implementation of the i4Q$^{DR}$, according to the specifications provided in deliverables D1.9 [3] and D2.7 [2], will be available at M24.

More specifically, this section is structured as follows. First, Section 3.1, describes what has been implemented so far, until M18. Then, Section 3.2 explains the actions that will be carried out to complete the implementation of the i4Q$^{DR}$ solution. Finally, Section 3.3 provides the history of the implementation.

## 3.1 Current implementation

The implementation of a solution such as the i4Q$^{DR}$ is challenging since, as explained above in Section 2.1., the i4Q$^{DR}$ solution has to be interoperable with most of the other solutions of the i4Q system and is involved in all pilots. This means that the implementation of the i4Q$^{DR}$ must be flexible enough to adapt to different scenarios and technical requirements but, at the same time, approach them in the most uniform way possible. In *"D3.7 i4Q$^{DR}$ Guidelines for Building Data Repositories for Industry 4.0."*, due on M18, we will present a set of recommendations and design specifications that can be followed to address these challenges and requirements. The implementation of the i4Q$^{DR}$ is driven by the guidelines that will be described in D3.7 [4].

The implementation of the i4Q$^{DR}$ has been organised in two phases. In the first one, the goal is to provide the solution's core features. In this regard, the plan is to develop a *collection of toolkits*, that is, a set of software artifacts, to configure, bootstrap and manage a number of tools and technologies related to the storage and management of data, supporting different usage scenarios. The purpose of these toolkits is to provide *automatable* mechanisms to configure, deploy, manage, diagnose and undeploy the tools and technologies. Such toolkits provide a basic version of the features described in Section 1.2.

The second phase of the implementation is aimed at providing a more advanced version of the solution. Not only by providing all the features described in Section 1.2., but also by providing them in a more refined way. For instance, in this phase we plan to offer a more unified interface to these toolkits.

Currently, the first implementation phase has almost been completed since we have finished the development of most of the above-mentioned toolkits. Regarding the second implementation phase, we have performed some preliminary works. The rest of this section is devoted to the current status of the implementation, which falls mostly into the first phase, whereas the second phase is described in more detail in Section3.2, where we explain the implementation's next steps

Regarding the first phase of the implementation of the i4Q$^{DR}$, we first gathered the requirements from the pilots and the rest of i4Q solutions interacting with the i4Q$^{DR}$, in order to select the storage technologies and to identify the different usage scenarios that had to be considered. In this document we briefly explain the storage technologies and usage scenarios that have been considered so far for the i4Q$^{DR}$. The decision-making process and the rationale followed to make such selections will be described in detail in deliverable D3.7 [4] (Section 4), as an illustrative example of the guidelines on how to build and design data repositories for Industry 4.0 that will be provided in that document.

With respect to the storage technologies that will be supported by the i4Q$^{DR}$, we have selected several of them to and satisfy the requirements of pilots and i4Q solutions and cover different types of data storage, namely:

Cassandra[2], a wide-column NoSQL distributed database server, appropriate for managing massive amounts of data.

- *MariaDB*[3], a SQL relational database server, very similar to MySQL.
- *MinIO*[4], which offers a high-performance, S3 compatible object storage. It is used to store files and is compatible with any public cloud.
- *MongoDB*[5], a JSON document-oriented database server.
- *MySQL*[6], a SQL relational database server.
- *Neo4J*[7], a graph database server that allows storing data relationships.
- *PostgreSQL*[8], a SQL relational database server.
- *Redis*[9].an in-memory data structure store, used as a distributed, in-memory key–value database, cache, and message broker, with optional durability.

Concerning the usage scenarios, we defined four of them, addressing different needs in terms of computing resources and security features. Namely*: "Single Server"* (SS), *"High Availability"* (HA), *"Single Server with Security"* (SS+Sec), and *"High Availability with Security"* (HA+Sec). The "Single Server" scenarios correspond to setups in which only one instance of the storage technology needs to be deployed, whereas the "High Availability" scenarios are suitable for cases with higher requirements in terms of computing resources and fault-tolerance, where cluster or replica set environments are more adequate. Finally, the "with Security" scenarios refer to configurations involving the TLS protocols, mostly for using x509 certificates, whereas the regular ones do not include such a feature and, thus, are only recommended for development and experimental purposes.

After performing the selection explained above the goal was to implement several toolkits, each one deploying one of selected storage technologies for each one of the defined scenarios. That is, a toolkit deploying Cassandra for the Single Server scenario, another one deploying Cassandra for the High Availability scenario, and so on. Some of these toolkits have already been implemented and their source code is available at the GitLab's private repository containing the source code of i4Q$^{DR}$ at: https://gitlab.com/i4q/dr. Although each toolkit is closely related to the

2 Cassandra. https://cassandra.apache.org
3 MariaDB. https://mariadb.org
4 MinIO. https://www.min.io
5 MongoDB. https://www.mongodb.com
6 MySQL. https://www.mysql.com
7 Neo4J. https://www.neo4j.com
8 PostgreSQL. https://www.postgresql.org
9 Redis. https://redis.io

tools or technologies it handles, all the ones developed so far share a number of principles and characteristics:

- *Human-friendly and automatable configuration*. The toolkits can be configured by means of automatable artifacts like configuration files and environment variables. These artifacts are designed to be easily managed by human users as well as by external tools like shell scripts, automation tools, etc.
- *Human-friendly and automatable operation*. Each toolkit provides one *main shell script* per scenario, so each scenario can be easily bootstrapped by a human user. Moreover, the toolkits are provided in the form of Bash script files that offer a number of functions, which are specific to a given scenario, a given tool or technology or are generic. Thus, they can easily be used in a fully automatable fashion. For instance, they can be integrated in other Bash scripts, used by continuous integration tools, etc.
- *Independence among toolkits*. Each toolkit can be used independently from other toolkits.

More specifically, each one of these toolkits *builds* and *launches* one or more *Docker containers* containing an instance of the corresponding storage technology. In the case of the "Single Server" scenario, only one instance is deployed, whereas more than one are deployed in the case of the "High Availability" scenario. Note, however, that toolkits for any type of scenario may build and run containers for other tools and purposes. For instance, the MongoDB toolkit also deploys a container to run an instance of Mongo Express, a web-based interface to administrate MongoDB instances. The "with Security" scenarios involve the use of the TLS protocol, so that each node uses TLS certificates for authentication purposes when interacting among themselves.

Moreover, the toolkits implemented so far share a similar structure, and consist, mainly, of three sub-components:

- The *storage configuration file*, specifying the value of some properties of the storage technology that can be manually set by the user.
- The *orchestration configuration files*, in which the configuration and properties of the corresponding Docker container(s) are declared.
- A set of *executable files*, which automatically execute all the necessary functions to build and run the corresponding Docker container(s), in a transparent way to the user.

These sub-components allow a potential user to:

- Configure the tool or technology it handles, by means of configuration files and environment variables.
- Prepare the local filesystem (for instance, the directories shared between the local host and the Docker containers).
- Start the Docker containers, provision and configure them and start in them whatever servers that are needed.
- Check the status of the tools and technologies deployed.
- Undeploy and dispose the Docker containers (and the data managed by them, if required).

The efforts to develop the different toolkits have been distributed among the different task partners. Table 1 gathers, for each storage technology ("Storage Tech" column) and scenario (second column), which partner is responsible for the implementation of the corresponding toolkit (see columns "Responsible") and its status by the time of writing this deliverable, which is specified by the "Status" column. More specifically, we consider three possible values for the "Status" column, namely:

- *"Pending":* denoting that the implementation of this toolkit has not started yet.
- *"In progress":* indicating that the development of this toolkit has started but is not ready yet.
- *"Done":* showing that the development of the toolkit has finished, and the source code is available at the GitLab repository.

| Storage Tech | Scenario | Responsible | Status |
|---|---|---|---|
| Cassandra | SS | ITI | Done |
| | HA | ITI | Pending |
| | SS+Sec | ITI | Pending |
| | HA+Sec | ITI | Pending |
| MariaDB | SS | ENG | Done |
| | HA | ENG | Pending |
| | SS+Sec | ENG | In Progress |
| | HA+Sec | ENG | Pending |
| MinIO | SS | ITI | Done |
| | HA | ITI | Done |
| | SS+Sec | KBZ | Done |
| | HA+Sec | KBZ | In Progress |
| MongoDB | SS | ITI | Done |
| | HA | ITI | Done |
| | SS+Sec | ITI | Done |
| | HA+Sec | ITI | Done |
| MySQL | SS | ITI | Done |
| | HA | ENG | Done |
| | SS+Sec | ITI | Done |
| | HA+Sec | ENG | Done |
| Neo4j | SS | ITI | Done |
| | HA | ITI | Pending |
| | SS+Sec | ITI | Done |
| | HA+Sec | ITI | Pending |
| PostgreSQL | SS | UNI | Done |
| | HA | UNI | Pending |

| | | | |
|---|---|---|---|
| | SS+Sec | UNI | In Progress |
| | HA+Sec | UNI | Pending |
| **Redis** | SS | ITI | Done |
| | HA | UNI | Pending |
| | SS+Sec | ITI | Done |
| | HA+Sec | UNI | Pending |

**Table 1.** Summary of toolkits' implementation status

Table 1 shows the implementation status "Pending" for several toolkits. There are several reasons why these developments have not started yet. In the case of the toolkit for Cassandra for scenarios other than "Single Server", or the Neo4j toolkit for the "High Availability" scenarios (regular and with security) the reason of reporting this status is because currently it is unclear whether such toolkits are actually required by another solution or pilot or, at least, it is not a requirement for the pilots' demonstrations that must be ready by M18 and, thus, have a lower priority than other developments. The development of the "High Availability" scenarios (regular and with security) of the MariaDB toolkit is pending because MySQL provides similar features. Indeed, from the technical point of view, MariaDB is what developers call a "fork" of the MySQL Project and, thus, have a common foundation and share most of their features. Furthermore, in this case we are also waiting for the results of the preliminary experiments performed as part of the second implementation phase, to decide the best and most efficient implementation approach.

### 3.1.1 Solution features analysed and mapping with user requirements

A set of features has already been developed for i4Q$^{DR}$, based on the set of user requirements referring to i4Q$^{DR}$ [3] and in line with the functional viewpoints [5]. Similar requirements have been assigned into common categories of tasks based on an extensive technical study conducted on user requirements, available datasets, etc., introduced to ensure the generalization abilities of the i4Q$^{DR}$ solution.

In the following, we explain in more detail which solution's features address and cover each one of the requirements defined in D1.9 [3]for the i4Q$^{DR}$.

- *PC1r2.4 "Capability of storing data for future retrieval and analysis":* is covered by the feature of being able to save image blob and structured data into a database, so that it can be retrieved later for its analysis.
- *PC1r3.2 "Capability of extracting relevant (requested) features from ingested data/signals":* which is supported by the fact that the i4Q$^{DR}$ can execute a given query to retrieve the desired data from the corresponding storage technology/tool.
- *PC2r3 "I4Q - DATA REPOSITORY: Define the right data repository":* is covered by the feature of being able to save structured data.
- PC4r1.1.2 *"Gather information from human source and take it into account for analysis":* is covered by the solution's features allowing to import data from a database and being able to save image blob and structured data.
- *PC4r4.2 "Possibility of configure the data taking process":* is covered by the solution's features that enable the management of both image blob and structure data.

- PC4r4.1 *"Extract valuable information from existing data and make it as an input for other exploitation or analysis"*: can be achieved by combining several features of the solution. More specifically, by: (i) importing/exporting data from/to a database, (ii) executing the appropriate queries on top of the stored data, and (iii) saving new data or updating previously stored data. These features refer to the two main types of data considered by this solution: image blobs and structured data.
- PC4r5.1.1 *"Avoid loses and corruption of data due to communication failure":* is covered by the feature that allows creating several replicas to store data and saving both image blob and structured data.
- PC4r5.4.1 *"Extract valuable info from existing data"*: is covered by the solution's capabilities to import data from an existing database and running queries on it.
- PC6r8.2 *"Dataflow from machines to database shall be established":* is mainly supported by the features that allow one to (i) manage the data repository so that it can receive the information provided by other components (e.g., other tools or solutions) as input, and (ii) save into a database image blobs.
- PC6r8.3 *"Image Data Compression":* is covered by the features of saving and updating image blobs.

Furthermore, there are several requirements that are supported by the capability of the solution to save structured data and image blobs, namely:

- PC3r3.1 "Prediction Result Storage"
- PC3r3.2 "Importance and post analytical Storage"
- PC4r1.1.1 "Disponibility[10] of production data taking from the CNC program and production orders"
- PC4r1.1.3 "Current measuring machines data gathering"
- PC4r1.3 "Store the data to be used in future purposes"
- PC4r5.3 "Collect data from status sensors (time-stamped)"
- PC5r3.3 "Data storage"
- PC6r1.4.1 *"Store the data",* which refers to the storage of injection machine parameters.
- PC6r1.4.2 *"Store the data (2)"*, which stands for the storage of energy analyser parameters.
- PC6r1.4.3 *"Store the data (3)",* which refers to the storage of water pump parameters.
- PC6r8.1 "Historical image data should be managed in the data repository".

## 3.2 Next developments

In order to complete the full implementation of the i4Q[DR], several actions have been planned as future work. First of all, we will finish the development of the toolkits whose implementation status in Table 1 is "Pending" or "In progress", which will complete the first phase of implementation.

Then, we will start the second implementation phase, for which we have defined actions that will be taken. On the one hand, we plan to improve the implementation of the "High Availability" scenario. Currently, the different replicas of the storage tool instance run in the same machine.

[10] In the sense of "availability"

However, a more realistic and resilient implementation would be one in which each replica is deployed in a different (virtual) machine. This can be achieved by using Docker Swarm[11].

On the other hand, we plan to implement a layer on top of the different toolkits. The goal of this layer is two-fold. Firstly, it will improve the interoperability of the i4Q^DR with the rest of the solutions, by offering a common interface for any of the toolkits. Secondly, it will ease the support to more storage technologies by the i4Q^DR if necessary in the future, which is a need that have been identified along the first months of the i4Q solutions development. In the following, we will refer to this layer as *"top-layer"*.

For the implementation of the top-layer we plan to use Trino[12] which is a highly parallel and distributed ANSI SQL-compliant open-source query engine that offers a relational-like view of different data storage tools. Moreover, Trino allows the execution of *federated queries,* which means that several databases of different types (relational, object storage, streaming or NoSQL, etc.) can be accessed within the same query.

Trino fulfils the goals of the top-layer as follows. In the one hand, Trino offers a REST API which will allow the interaction of the i4Q^DR with other solutions by means of the HTTP protocol. Furthermore, there is a Python client package that enables the implementation of client applications connecting to Trino's servers. This package can be used to develop a subcomponent facilitating the interoperability of the top-layer with other solutions. On the other hand, Trino facilitates the integration of new storage technologies via the so-called "connectors". Basically, a connector is a piece of software that adapts Trino to a data source, as if it was a driver for a database[13]. Trino contains several built-in connectors and many third-party have contributed connectors for other technologies. The list of currently available connectors is provided at: https://trino.io/docs/current/connector.html. This list includes connectors for all the storage technologies mentioned in Section 3.1 except MinIO and Neo4j. For these two cases, we have to search for and analyse possible alternatives. However, we believe that the use of Trino brings enough benefits to consider its use in the implementation of the i4Q^DR whenever possible, even though it does not support all the selected storage technologies.

We have already performed some preliminary works regarding the implementation of the top-layer, to integrate some toolkits already developed into Trino. More specifically, ENG started testing the Single Server scenario of both MariaDB and MySQL on Trino, whereas ITI has successfully done so for the Single Server scenario with MongoDB. We will continue to work on this direction to make this integration more configurable and complete the integration of the other toolkits.

Another line of work we will explore after M18 is the enhancement of the interoperability of the i4Q^DR with other solutions of the i4Q system, especially with the i4Q Message Bus. Firstly, we will explore Trino's REST API and the Python package to implement client applications, as explained above. Then, we will also analyse whether we need to implement more sophisticated methods to

[11] See https://docs.docker.com/engine/swarm/ for further information on how to use Docker in swarm mode.

[12] https://trino.io/

[13] https://trino.io/docs/current/overview/concepts.html, see "Connector" subsection.

import/export data into/from the i4Q$^{DR}$, other than the ones provided by the storage technologies supported by the different toolkits.

Finally, we will explore in more detail the implementation of the features related to enhance the security of the solution. In this sense, we plan to improve the "with Security" scenarios to have a more realistic implementation. For instance, we will explore whether the TLS certificates can be generated by the i4Q$^{SH}$ solution. Furthermore, we will study the best approach to incorporate into the i4Q$^{DR}$ an access control mechanism.

## 3.3 History

This section provides the version history of the i4Q$^{DR}$ implementation up to M18, which is gathered in the table below. More specifically, it shows, for each version of the implementation (denoted by the first column), when it was released (see "Release date" column), and what functionality was added (described in column "New features").

| Version | Release date | New features |
|---|---|---|
| V0.0.1 | 21/01/2022 | Added toolkit for MongoDB (SS and HA scenarios) |
| V0.0.2 | 24/01/2022 | Added toolkit for MySQL (SS, SS+Sec scenarios), and preliminary version of tookit for Redis (SS scenario) |
| V0.0.3 | 25/01/2022 | Added SS+Sec scenario to Redis tookit. Included technical improvements. |
| V0.0.4 | 26/01/2022 | Added toolkit for MinIO (SS scenario) |
| V0.0.5 | 27/01/2022 | Applied minor changes and refactoring, and added documentation about the tookits' commons. |
| V0.0.6 | 28/01/2022 | Added tookit for Cassandra (SS), and applied minor technical improvements |
| V0.0.7 | 31/01/2022 | Added HA scenario to MinIO toolkit |
| V0.0.8 | 01/02/2022 | Added toolkit for Neo4j (SS scenario) |
| V0.0.9 | 02/02/2022 | Added scenario SS+Sec to Neo4j toolkit |
| V0.0.10 | 15/02/2022 | Updated MongoDB Docker image to v5.0.6, added SSL initialization, improved MongoDB HA scenario, and applied other technical improvements. |
| V0.0.11 | 30/04/2022 | Added PostgreSQL toolkit (SS scenario) |
| V0.0.12 | 06/05/2022 | Added preliminary integration of MongoDB toolkit for SS scenario with Trino |
| V0.0.13 | 04/05/2022 | Added toolkit for MariaDB (SS scenario) |
| V0.0.14 | 11/05/2022 | Added preliminary integration of MariaDB and MySQL toolkits for SS scenario with Trino |
| V0.1 | 30/05/2022 | M18 solution release |

**Table 2.** i4Q$^{DR}$ Version history

# 4. Conclusions

Deliverable *"D3.8 - i4Q Data Repository"* is a technical specification document presenting a technical overview of the i4Q Data Repository solution (i4Q$^{DR}$). In this deliverable we have described in detail the role, the functionalities, and the conceptual architecture of i4Q$^{DR}$.

Moreover, we have explained the main features of the solution that will be available by the end of the task in M24, describing its architecture diagram with respect to i4Q Reference Architecture.

Furthermore, in this document we have explained in detail the implementation work of the i4Q$^{DR}$ solution. We first provided an overview of the approach we are following and described the status of the implementation up to M18, explaining which functionalities of the i4Q$^{DR}$ have been developed so far. In this regard, we included the analysis and engineering of the pilots' requirements for this solution, to clarify the technical specifications.

Finally, we also provided a summary of what needs to be implemented until the end of the task in M24 in order to complete the implementation of the solution. These developments will be explained in detail in deliverable *"D3.16 - i4Q Data Repository v2"*, due on M24.

# References


[1] i4Q, *D4.1 – i4Q Data Integration and Transformation Services* (June 2022)

[2] i4Q, *D2.7 – i4Q Reference Architecture and Viewpoints Analysis v2*. (Sep 2021)

[3] i4Q, *D1.9 – Requirements Analysis and Functional Specification v2. (*Sep 2021)

[4] i4Q, *D3.7 – i4Q Guidelines for Building Data Repositories for Industry 4.0* (June 2022)

[5] i4Q, *D2.6 – Technical Specifications,* (Sep 2021)

# Appendix I

The PDF version of the **4Q Data Repository** (i4Q^DR) technical web documentation, which can be accessed online at: **http://i4q.upv.es/8_i4Q_DR/index.html**.

**i4Q Data Repository (i4Q^DR)**

**General Description**

The i4Q Data Repository (i4Q^DR) is a distributed storage system that will oversee receiving, storing, and serving the data in an appropriate way to other solutions. This solution is suitable to support and enhance a high degree of digitization in companies with most manufacturing devices acting as sensors or actuators and generating vast amounts of data.

The i4Q Data Repository (i4Q^DR) is aimed at providing, in a centralised fashion, the functionality related to the storage of data in the whole i4Q system. Indeed, the i4Q^DR is involved in all the pilots and is expected to interoperate with a large subset of the i4Q solutions.

The implementation of the i4Q^DR has been organised in two phases. In the first one, the goal is to provide the solution's core features. In this regard, the plan is to develop a collection of toolkits to configure, bootstrap and manage a number of tools and technologies related to the storage and management of data, supporting different usage scenarios. The purpose of these toolkits is to provide automatable mechanisms to configure, deploy, manage, diagnose and dispose the tools and technologies.

The second phase of the implementation is aimed at providing a more advanced version of the solution. For this purpose, we will implement a layer on top of the different toolkits aimed at offering a common interface with any of the toolkits. For this purpose, we plan to use Trino, a tool that, among other features, allows to execute federated queries in a very efficient fashion. "Federated queries" refer to the possibility of querying databases of different types (relational, object storage, streaming or NoSQL), all in the same query.

The first implementation phase has been almost completed. We have started this implementation phase very recently and have performed some preliminary works. More specifically, ENG started testing the Single Server scenario of both MariaDB and MySQL on Trino, whereas ITI has successfully done so for the Single Server scenario with MongoDB. We will continue to work on this direction to make this integration more configurable and complete the integration of the other toolkits. Therefore, this web documentation refers, mostly, to the first phase.

**Features**

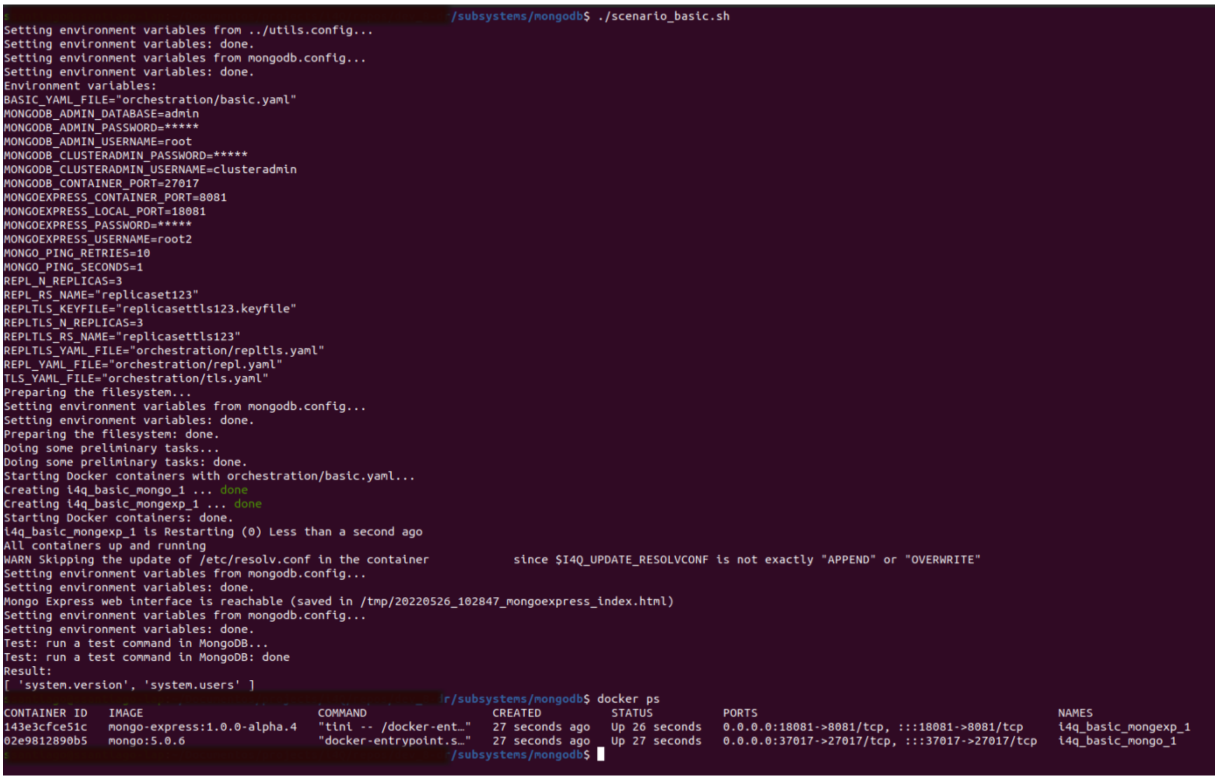
The features of the i4Q^DR are as follows:

1. **An access control mechanism**: to ensure that only authorised entities have access to the data and the related tools. Users with the appropriate permissions will be able to configure this access control, to grant access to the allowed entities.
2. **Tools and technologies to manage structured data**: this feature will be offered by means of DBMSs that may be relational SQL-based, document (e.g., JSON-based), general NoSQL tools, etc.

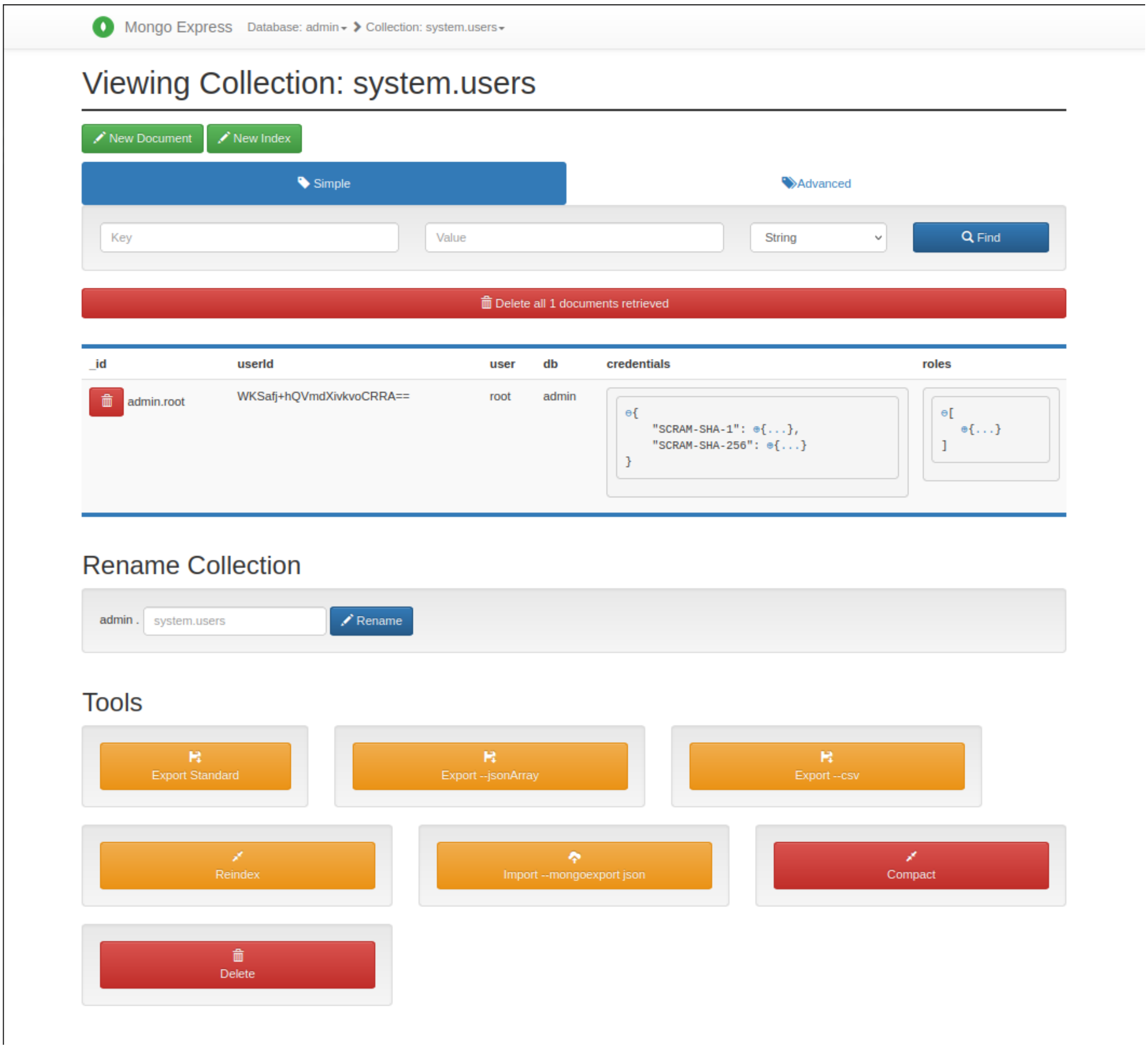
3. **Tools and technologies to manage blobs**: this feature will be offered through tools that offer support for blobs like some general-purpose DBMSs or even specific ones (e.g., Minio).
4. **Efficient mechanisms to query the stored data and retrieve the results of such queries**: for both structured data and blobs.
5. **Mechanisms to import/export data to/from the i4Q$^{DR}$**: o ease the interoperability of this solution with others.
6. **Mechanisms to manage the data repository itself**.

Note, however, that these features will be fully implemented in the final version of the i4Q$^{DR}$, due on M24.

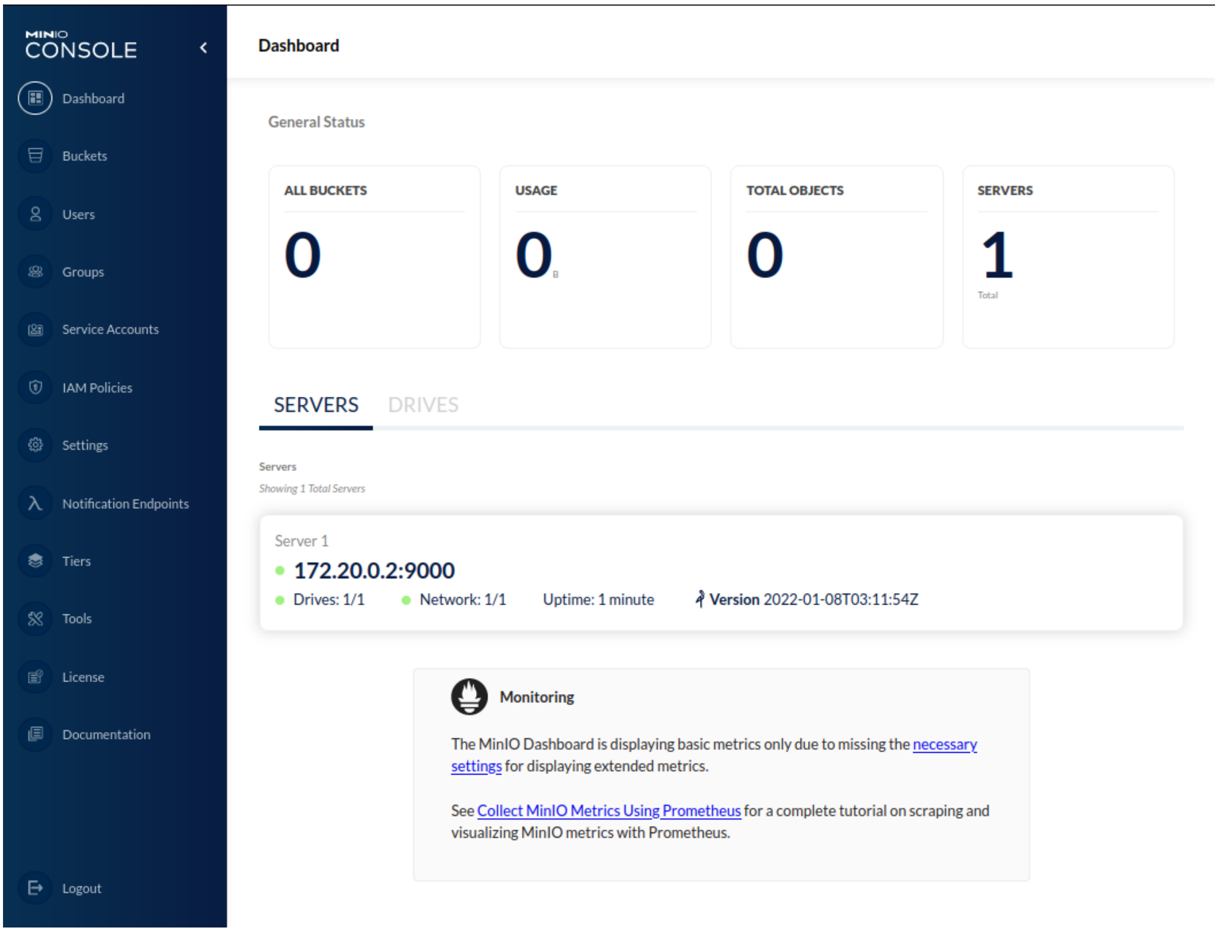
**ScreenShots**



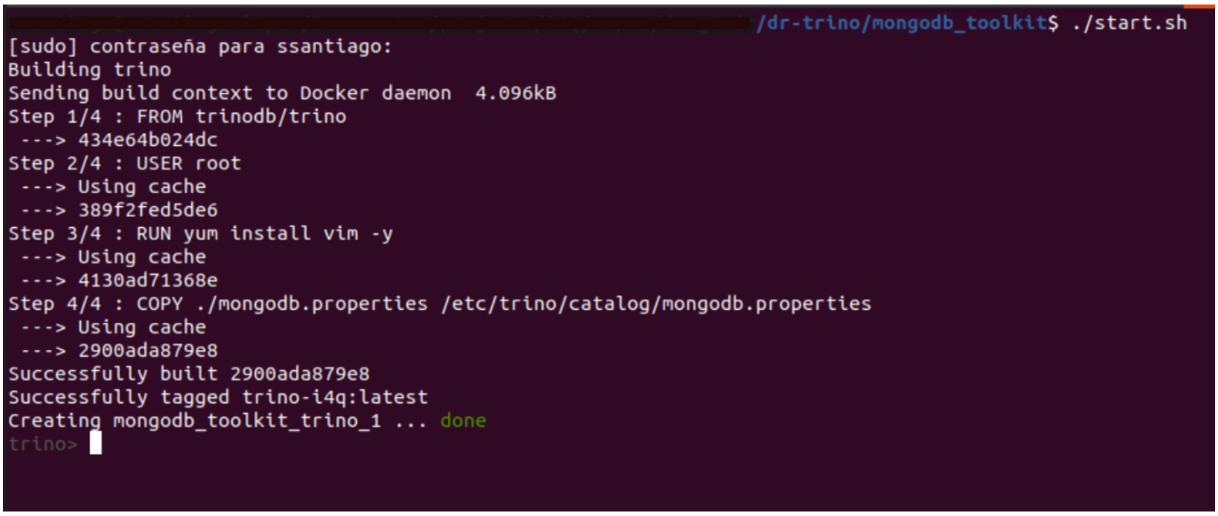

Bootstrapping single server scenario for MongoDB from console




MongoExpress instance running after bootstrapping SS scenario for MongoDB




MinIO instance running after bootstrapping SS scenario




Bootsrapping of Trino with a MongoDB connector

### Commercial Information

#### Authors

| Company | Website | Logo |
|---|---|---|
| ITI | https://www.iti.es | |
| Uninova | https://www.uninova.pt | |
| Engineering | https://www.eng.it | |
| CERTH | https://www.certh.gr | |
| Knowledgebiz | https://knowledgebiz.pt/ | |

#### License

- A free-software license

#### Pricing

| Subject | Value |
|---|---|
| Payment Model | One-off |
| Price | 0 € |

**Associated i4Q Solutions**

**Required**

Currently, it can operate without the need for another i4Q solution. However, it is expected to require the use of i4Q$^{SH}$ solution for SSL certificates.

**Optional**

None. However, the i4Q$^{DR}$ is expected to be used with almost any other i4Q solution in the pilots, either to store data, or to allow the retrieval of previously stored data.

**System Requirements**

Docker requirements:

- 4 GB Ram
- 64-bit operating system
- Hardware virtualisation support

Additionally, it requires the following dependencies to be already installed:

1. *Bash*, to run the scripts. They contain a few syntax details specific to Bash that may not work with other shells (Dash, csh, ksh, zsh, etc.). Any recent version will do.
2. *Docker* and *Docker Compose*, to deploy and run containers. Any recent version accepting version 3.9 Docker Compose YAML files will do.
3. *OpenSSL*, to create test SSL artifacts (keys, certificates, etc.). Any recent version will do.
4. *curl*, to retrieve files required to build the Docker images.
5. *jq*, to manipulate JSON files. Any recent version will do. This is required by the toolkits for MinIO and MongoDB.

**API Specification**

Since Trino is expected to be deployed as a layer on top of the different toolkits, the plan is to use Trino REST API as a mechanism to interact with the i4Q$^{DR}$. This REST API offers the following endpoints:

| Resource | POST | GET | PUT | DELETE |
|---|---|---|---|---|
| /v1/statement | runs the query string in the *POST*` body, and returns a JSON document containing the query results. If there are more results, the JSON document contains a *nextUri* URL attribute. | Not Supported | Not Supported | Not Supported |
| */nextUri*` | Not Supported | Returns the next batch | Not Supported | Terminates a running query |

| Resource | POST | GET | PUT | DELETE |
|---|---|---|---|---|
| | | of query results | | |

**Installation Guidelines**

| Resource | Location |
|---|---|
| Last release (v.0.1.0) | Link |
| Video | TBD |

**Common bootstrapping of toolkits' scenarios**

This section provides general information on how to bootstrap and use the different toolkits. For further information and specific details, we refer the reader to the i4QDR GitLab repository.

The general way to use a toolkit consists in bootstrapping one of the scenarios it offers.

For instance, to bootstrap the *basic* scenario for the *mongodb* toolkit, do this:

```
$ cd mongodb/
$ ./scenario_basic.sh
```

The result of bootstrapping a scenario is usually a number of Docker containers started and running and one or more server or services made available. For instance, after starting the *basic* scenario for the *mongodb* toolkit, there is an *i4q_basic_mongo_1* container running, that hosts a regular MongoDB server.

Then, a number of actions can be performed on the scenario to manage the containers in different ways. To perform them, it is necessary to first load the script that implements the toolkit.

**Bootsrapping of Trino with connector for MongoDB**

As explained above, we have performed some experiments deploying Trino on top of the toolkits. More specifically, we have deployed Trino on top of the MongoDB toolkit for the single server scenario. To bootstrap this scenario, it is necessary to run the following commands in the command line:

```
$ cd dr-trino/mongodb_toolkit/
$ ./start.sh
```

**User Manual**

This section provides general information on how to use the different toolkits. For further information and specific details, we refer the reader to the [i4QDR GitLab repository](#).

**Usage of toolkit-related and scenario-related functions**

The set of toolkit-specific functions varies from one toolkit to another but generally speaking, there are a number of functions that are offered by many if not all the toolkits, which are of interest during the lifetime of the scenario.

The following examples show how to use them, for a given scenario of a given toolkit.

First, bootstrap the scenario:

```
$ cd subsystems/minio/
$ ./scenario_basic.sh
...
```

Then, load the corresponding toolkit functions:

```
$ source ./_functions_minio.sh
```

Load the constants related to the current toolkit:

```
$ i4q_minio_env
Setting environment variables from minio.config...
Setting environment variables: done.
```

Print the constants related to the current toolkit. Note that constants whose name include "PASSWORD" are hidden:

```
$ i4q_minio_printenv
Environment variables:
BASIC_YAML_FILE="orchestration/basic.yaml"
...
MINIO_ROOT_PASSWORD=*****
MINIO_ROOT_USER=minio_admin
```

Moreover, each toolkit may offer additional *i4q_<TOOLKIT>_** toolkit-related functions. See the toolkit's *README.md* for additional information.

Show the running Docker containers of the scenario:

```
$ i4q_basic_ps
      Name                  Command         State         Ports
---------------------------------------------------------------
i4q_basic_minio_1   /usr/bin/docker-   Up      0.0.0.0:19000->9
                    entrypoint ...             000/tcp,...
```

Show the logs of the running Docker containers of the scenario:

```
$ i4q_basic_logs
...
i4q_basic_minio_1 | API: http://192.168.64.2:9000  http://127.0.0.1:9000
i4q_basic_minio_1 | Console: http://192.168.64.2:9001 http://127.0.0.1:9001
i4q_basic_minio_1 | Documentation: https://docs.min.io
^C
```

Start a shell in one of the Docker containers of the scenario:

```
$ i4q_basic_shell i4q_basic_minio_1
[root@i4q_basic_minio_1 /]# ...
[root@i4q_basic_minio_1 /]# exit
```

Stop the Docker containers of the scenario:

```
$ i4q_basic_stop
Stopping i4q_basic_minio_1 ... done
$ i4q_basic_ps
      Name                   Command              State    Ports
-------------------------------------------------------------
i4q_basic_minio_1   /usr/bin/docker-entrypoint   Exit 0
                    ...
```

Remove the Docker containers of the scenario:

```
$ i4q_basic_rmcont
Going to remove i4q_basic_minio_1
Removing i4q_basic_minio_1 ... done
$ i4q_basic_ps
Name   Command   State   Ports
------------------------------
```

Remove the data managed by the Docker containers in the scenario (use with caution!):

```
$ i4q_basic_clean
```

Moreover, each toolkit may offer additional *i4q_<S>_** scenario-related functions. See the toolkit's *README.md* for additional information.

**Starting over**

Sometimes, it is necessary to start over the bootstrapping of a scenario. This is especially relevant in some situations such as performing debugging tasks, using it for the first time, testing new functionality, etc.

The following example bootstraps a given scenario of a given toolkit, undeploys it and prepares the environment to start over:

```
$ cd minio/
$ ./scenario_basic.sh
...
$ i4q_minio_stop && i4q_minio_rmcont && i4q_minio_clean
```