



D5.2 - Database and Data- Control Bus Documentation

**WP5 - Water Cyber Physical System: The Industrial
Water Efficiency Management System**

Publish Date: April 2024

Authors: Aziz Mousas, Georgios V. Lioudakis, Kostas Kalaboukas, Mariza Koukovini,
Nikolaos Dellas, Eugenia Papagiannakopoulou



The AquaSPICE project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 958396.

Document Information

GRANT AGREEMENT NUMBER	958396	ACRONYM	AquaSPICE
FULL TITLE	Advancing Sustainability of Process Industries through Digital and Circular Water Use Innovations		
START DATE	1 st December 2020	DURATION	51 months
PROJECT URL	www.aquaspice.eu		
DELIVERABLE	D5.2 – Database and Data-Control Bus Documentation		
WORK PACKAGE	WP5 – Water Cyber Physical System: The Industrial Water Efficiency Management System		
DATE OF DELIVERY	CONTRACTUAL	07/2022	ACTUAL 04/2024
NATURE	Report	DISSEMINATION LEVEL	Public
LEAD BENEFICIARY	Maggioli Spa		
RESPONSIBLE AUTHOR	Aziz Mousas, Kostas Kalaboukas		
CONTRIBUTIONS FROM	Georgios V. Lioudakis, Mariza Koukovini, Nikolaos Dellas, Eugenia Papagiannakopoulou, Robert Sanfeliu Prat, Dimitris Apostolou		
ABSTRACT	<p>This deliverable provides an overview of the mechanisms offered by the WaterCPS platform regarding data ingestion, service integration that enable the digital twins paradigm. Based on D5.1 “<i>WaterCPS Services Specifications, Functional Model and System Design</i>”, this deliverable specifies the overall data and communication model for the platform and apply that to implement the ICT substructure required to ascertain data availability and facilitate data flows. A complete set of data services are specified and implemented to satisfy all pilot cases requirements, service requirements and functions specified by D5.1. Finally, a data communication and control bus is specified and designed to connect the database and all components, enabling data flows and communication.</p>		

Document History

VERSION	ISSUE DATE	STAGE	DESCRIPTION	CONTRIBUTOR
0.1	15 Jul 2022	First version	First version of the document, prior to internal AquaSPICE members review.	Aziz Mousas, Kostas Kalaboukas, Georgios V. Lioudakis, Mariza Koukovini, Nikolaos Dellas, Eugenia Papagiannakopoulou
0.2	22 Jul 2022	Final version	Final version after review	Aziz Mousas, Georgios Lioudakis, Kostas Kalaboukas, Robert Sanfeliu Prat, Dimitris Apostolou
0.3	29 Jul 2022	Final version	Submission	
0.4	24 Mar 2023	Updated version	Deliverable updates to accommodate comments	Aziz Mousas, Kostas Kalaboukas
0.5	16 Apr 2024	Updated version	Deliverable updates to accommodate comments	Aziz Mousas, Kostas Kalaboukas

Disclaimer

Any dissemination of results reflects only the author's view and the European Commission is not responsible for any use that may be made of the information it contains.

Copyright message

© AquaSPICE Consortium, 2022-2024

This deliverable contains original unpublished work except where indicated otherwise. Acknowledgement of previously published material and of the work of others has been made through appropriate citation, quotation, or both. Reproduction is authorised if the source is acknowledged.

TABLE OF CONTENTS

1.	Executive summary.....	7
2.	Introduction.....	9
2.1.	Purpose and scope.....	9
2.2.	Relation with other deliverables.....	9
2.3.	Relation with background knowledge.....	9
2.4.	Structure of the document.....	11
2.5.	Document updates.....	11
3.	AquaSPICE Database and Data-Control Bus.....	12
3.1.	Architecture overview.....	12
3.2.	Connection to the Real-Time Monitoring platform.....	13
4.	Configuring the AquaSPICE Database and Data-Control Bus for the case studies.....	16
4.1.	Methodology.....	16
4.2.	Deployment.....	16
4.3.	Data ingestion layer.....	17
4.3.1.	Eclipse Hono.....	17
4.3.2.	Apache NiFi.....	17
4.4.	Integration bus layer.....	20
4.4.1.	Apache ActiveMQ.....	20
4.4.2.	Apache Drill.....	22
4.5.	Digital Twins layer.....	23
4.5.1.	Eclipse Ditto.....	23
4.5.2.	Data model.....	26
5.	Conclusions.....	30
6.	References.....	31
7.	Annex: Open source technology stack overview.....	32
7.1.	Eclipse Hono.....	32
7.2.	Apache NiFi.....	33
7.3.	Apache ActiveMQ.....	37
7.4.	Apache Drill.....	38
7.5.	Eclipse Ditto.....	39

LIST OF FIGURES

Figure 1 – The database and data-control bus open source technology stack.....	12
Figure 2 – Connection of the RTM to the WaterCPS platform.....	14
Figure 3 – RTM platform subscription model	15
Figure 4 – RTM platform data query model.....	15
Figure 5 – Database and Data-Control Bus Configuration Workflow.....	16
Figure 6 – Database and data-control bus deployment.....	17
Figure 7 – Apache NiFi – First level configuration for a case study.....	18
Figure 8 - RTM-WaterCPS subscription for CS #3	19
Figure 9 – RTM notification example	19
Figure 10 - Ingesting RTM notifications and updating digital twins	19
Figure 11 - RTM-DT platform data mapping	20
Figure 12 - ActiveMQ message queues supporting CS#3	21
Figure 13 - Apache ActiveMQ message queues interconnection	21
Figure 14 - Apache Drill storage plugins	22
Figure 15 - Integration dataflow handling timeseries queries to the RTM	23
Figure 16 - Eclipse Ditto REST API	24
Figure 17 - Eclipse Ditto - Connection to the integration Bus.....	25
Figure 18 - Eclipse Ditto - Message routing dataflow	26
Figure 19 - Digital Twins Data Model - Static Information	27
Figure 20 - Digital Twins Data Model – Synchronization Feature	28
Figure 21 - Digital Twins Data Model - Value-Added Service feature	29
Figure 22 – Eclipse Hono overview	32
Figure 23 – Eclipse Hono architecture	33
Figure 24 – Apache NiFi architecture.....	34
Figure 25 – Data connector simple example	36
Figure 26 – EvaluateJSONPath NiFi processor properties.....	36
Figure 27 – ExecuteSQL NiFi processor properties	37
Figure 28 – Apache ActiveMQ - Publish/Subscribe.....	38
Figure 29 – Apache Drill interfaces	38
Figure 30 – Apache Drill - Drillbit architecture	39
Figure 31 – Apache Drill - Drillbit configuration.....	39
Figure 32 – Eclipse Ditto – Twin channel	40
Figure 33 – Eclipse Ditto – Live Channel	41
Figure 34 – Eclipse Ditto – Components view.....	43
Figure 35 – Eclipse Ditto - Class diagram of Ditto's most basic entities in API version 2.	44

ABBREVIATIONS/ACRONYMS

AMQP	Advanced Message Queuing Protocol
API	Application Programming Interface
CS	Case Study
DB	Database
dLCA	dynamic Life Cycle Assessment
DT	Digital Twin
EIP	Enterprise Integration Patterns
HTTP	HyperText Transfer Protocol
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
ICT	Information and Communication Technologies
IoT	Internet Of Things
JSON	JavaScript Object Notation
JSON-LD	JSON-Linked Data
MQTT	Message Queuing Telemetry Transport
OAuth	Open Authorization
OME	Observable Manufacturing Element
PSM	Process Simulation and Modelling
REST	Representational State Transfer
RQL	Resource Query Language
RTM	Real Time Monitoring
SQL	Structured query language
WaterCPS	Water Cyber Physical System

1. Executive summary

The aim of WP5 is to implement the entire AquaSPICE architecture and the system that integrates the WaterCPS, the Real-Time Monitoring platform and all tools and services over a cloud-based collaboration infrastructure, along the specifications reported in deliverable D5.1 “*WaterCPS Services Specifications, Functional Model and System Design*” [5].

A key prerequisite in this direction is the development of a mediation middleware among the various components comprising the AquaSPICE ecosystem, assigned with the interaction, coordination and orchestration of its components and operations. To address the above, AquaSPICE proposes a database and data-control bus framework offering the following functionalities:

- **Data acquisition:** It provides for the integration of the infrastructure objects and data sources, through a unified solution for accessing information stored in or originating from heterogeneous systems and the Real-Time Monitoring platform.
- **Messaging and streaming:** It facilitates both asynchronous and point-to-point message exchange between system components, while also supporting streaming, enabling on-the-fly and real-time processing of data as they arrive.
- **Digital twins as a single source of truth:** Digital twins constitute the sole point of reference regarding the state and behaviour of considered entities; in this sense, all related data are associated with the digital twins that represent the corresponding assets and are only retrieved from or through these digital twins, subject to their control.

In this setting, this deliverable presents the database and data-control bus framework, reflecting the work performed in the context of the project Task 5.2 “*Integration of Databases & Tools into WaterCPS*”, and the outcome thereof. Closely following the analysis of the pilot cases and the associated requirements derived, as well as the development of the rest of AquaSPICE components, services and tools, this deliverable describes the database, data-control and integration bus with the main functions of the underlying technologies that render them integration enablers in the AquaSPICE platform.

It has to be noted that the work presented in this deliverable is based on the background knowledge that is brought to the AquaSPICE project from the FACTLOG project. There, the authors were responsible to design and develop a digital twins platform focused on the manufacturing domain. In the context of the FACTLOG project the authors came up with an open source technology stack that is capable of supporting a fully fledged IoT platform, which is presented in the FACTLOG deliverable D6.2 “*Data Collection Framework (Final Version)*” [7].

The AquaSPICE Database and Data-Control Bus solution leverages the open source technology stack that was compiled initially for the FACTLOG project and applies it in the AquaSPICE domain.

The original version of the document has been revised following feedback from the project officer, who noted similarities to deliverable D6.2 of the FACTLOG project [7], and a lack of clear differentiation to the work performed therein.

In this updated version, the detailed technical descriptions of the open source technology stack that comprises the database and data-control bus and is shared with the FACTLOG project, has been relocated to the Annex [Chapter 7]. These technologies are developed and maintained by the Eclipse Foundation and the Apache Software Foundation.

Additionally, in order to improve clarity regarding the diverse contributions made on this premise during the AquaSPICE project, a section has been added to provide an overview of the advancements made beyond those in the FACTLOG project (Section 2.3).

Lastly, this version of the deliverable merges in a single chapter (Chapter 4) how AquaSPICE leverages the technology stack and goes one step further from the previous version, by providing additional illustrative examples and focusing on the configuration, setup and deployment of the database and data-control bus.

2. Introduction

2.1. Purpose and scope

The aim of WP5 is to implement the entire AquaSPICE architecture and the system that integrates the WaterCPS, the Real-Time Monitoring platform and all tools and services over a cloud-based collaboration infrastructure, along the specifications reported in deliverable D5.1 “WaterCPS Services Specifications, Functional Model and System Design” [5]. This deliverable reports mainly on the results of Task 5.2 “*Integration of Databases & Tools into WaterCPS*” presenting the database and data-control bus framework, which has been devised to support the AquaSPICE platform operation. For this task, careful analysis of pilot scenarios and data sources has taken place, as well as elaboration of use cases and system requirements.

2.2. Relation with other deliverables

This deliverable evolves the main data communication and integration principles set forth within the deliverable D5.1 and in this direction, it provides, on the one hand, the technical specifications meant to implement the Integration Bus functionalities in terms of data ingestion and management, and, on the other, some insights on how the digital twins platform deployed for AquaSPICE will make collected data accessible to all other interested applications and modules. Towards this goal, deliverables, D1.5 “Data Models, Taxonomy and Ontology Industrial Water Use” [2] and D3.2 “Data Management and Harmonization Plan” [4], provide the foundation for the specification of data models for the digital twins of each case study, while D3.1 “RTM Specification and System Architecture” [3] have been used as a starting point in identifying required dependencies and interactions between the WaterCPS and the Real-Time Monitoring Platform.

Certain aspects of the framework presented in this document will be elaborated in subsequent deliverables, namely D5.4/D5.5/D5.6 “WaterCPS Digital Platform prototype (1st / 2nd / Final). These deliverables will focus on the application of the framework to the AquaSPICE case studies and will validate the flexibility of the proposed database and data-control bus mechanisms in fulfilling their heterogeneous requirements. In addition, these deliverables will present any potential shortcomings of said mechanisms, along with guidelines and best practices for their configuration.

2.3. Relation with background knowledge

The work presented in this deliverable is based on the background knowledge that is brought to the AquaSPICE project from the FACTLOG project. There, the authors were responsible to design and develop a digital twins platform focused on the manufacturing domain. In the context of the FACTLOG project the authors came up with an open source technology stack that is capable of supporting a fully fledged IoT platform, which is presented in the FACTLOG deliverable D6.2 “Data Collection Framework (Final Version)” [7].

The AquaSPICE Database and Data-Control Bus solution leverages the open source technology stack that was compiled initially for the FACTLOG project and applies it in the AquaSPICE domain. These technologies are developed and maintained by the Eclipse Foundation and the Apache Software Foundation. Detailed technical descriptions of this stack can be found in the Annex (Chapter 7).

The table below summarises the advancements made to the database and data-control bus in the context of the AquaSPICE project, beyond those in the FACTLOG project, in order to adjust to the AquaSPICE use cases requirements.

Table 1: AquaSPICE contributions beyond the FACTLOG project

	FACTLOG	AquaSPICE
Architecture	<ul style="list-style-type: none"> 5 Layers (Application, Digital Twins, Integration, Ingestion, On premise) 	<ul style="list-style-type: none"> Device telemetry of the ingestion layer and telemetry storage of the on premise layer are covered by the RTM platform. (Section 3.2)
Digital Twins Layer	<ul style="list-style-type: none"> Eclipse Ditto v2.0.0 Based on ISO/DIS 23247-3 	<ul style="list-style-type: none"> Upgraded to Eclipse Ditto v2.4.1 Added support for Smart Data Ontology data model (Section 4.5.2)
Integration Bus Layer	<ul style="list-style-type: none"> Apache Drill v1.19.0 Apache ActiveMQ v2.13.0 	<ul style="list-style-type: none"> Upgraded to Apache ActiveMQ 2.23.1 FIWARE interoperability. NGSI v2-based timeseries queries (Quantum Leap) (Section 4.4.2)
Ingestion Layer	<ul style="list-style-type: none"> Apache NiFi v1.11.4 Eclipse Hono v1.6.0 Mongo DB v4.2 	<ul style="list-style-type: none"> Upgraded to Apache NiFi v1.13.2 Eclipse Hono is not used since RTM handles device telemetry FIWARE interoperability. NGSI v1-based event subscriptions (Orion Context Broker) (Section 4.3.2)
On premise Layer	<ul style="list-style-type: none"> Apache MiNiFi v0.5.0 	<ul style="list-style-type: none"> Apache MiNiFi is not used since RTM handles device telemetry (Section 3.2)
Deployment	<ul style="list-style-type: none"> Ubuntu v18.04.5 Docker v20.10.6 Manual proxy configuration 	<ul style="list-style-type: none"> Ubuntu v20.04.4 Added Apache Httpd v2.4 as proxy (Section 4.2) Proxy configuration as code

2.4. Structure of the document

The introduction first outlines what the purpose of this document is and the areas it covers, and then lists all the deliverables related to this document, serving either as its input or representing future work based upon its content, and the background knowledge that this work is based upon. An overview of the database and data-control bus framework is presented in Section 3. Then, Section 4 focuses on presenting how the database and data-control bus is setup, and provides illustrative examples on how each of its layers are configured. Section 4.1 and 4.2 present the development methodology and provide information on the deployment of the database and data-control bus. Then, starting with the ingestion layer the document presents the Eclipse Hono solution for the ingestion of IoT device data (Section 4.3.1), and the Apache Nifi for designing and executing data flows from legacy data sources (Section 4.3.2). In the integration bus layer, the Apache ActiveMQ is used as the message broker between platform components realizing the publish/subscribe communication paradigm, and for querying the attached to the platform data sources, Apache Drill is leveraged providing an SQL query interface. Section 4.5 presents the Digital Twins layer, where Eclipse Ditto is employed providing APIs for managing and interacting with digital twins. Finally, the Annex presents detailed technical descriptions of the open source technology stack that comprises the database and data-control bus.

2.5. Document updates

The original version of the document has been revised following feedback from the project officer, who noted similarities to deliverable D6.2 of the FACTLOG project [7], and a lack of clear differentiation to the work performed therein.

In this updated version, the detailed technical descriptions of the open source technology stack that comprises the database and data-control bus and is shared with the FACTLOG project, has been relocated to the Annex. These technologies are developed and maintained by the Eclipse Foundation and the Apache Software Foundation.

Additionally, in order to improve clarity regarding the diverse contributions made during the AquaSPICE project on this premise, a section has been added to provide an overview of the advancements made beyond those in the FACTLOG project (Section 2.3).

This version of the deliverable merges in a single chapter (Chapter 4) how AquaSPICE leverages the technology stack and goes one step further from the previous version, by providing additional illustrative examples and focusing on the configuration, setup and deployment of the database and data-control bus.

3. AquaSPICE Database and Data-Control Bus

Effective communication of data coming from the IoT devices and routing to the appropriate components is vital for the operation of AquaSPICE. To this end, the database and data-control bus facilitates the realization of the digital twins paradigm offering a toolset for connecting IoT devices and data sources to their digital representation, as well as APIs to applications leveraging digital twins to enhance their operation.

3.1. Architecture overview

The figure below presents the layered architecture of the AquaSPICE database and data-control bus, an open source technology stack originally compiled under the context of the FACTLOG project¹ and presented thoroughly in the Annex. These technologies are developed and maintained by the Eclipse Foundation and the Apache Software Foundation.

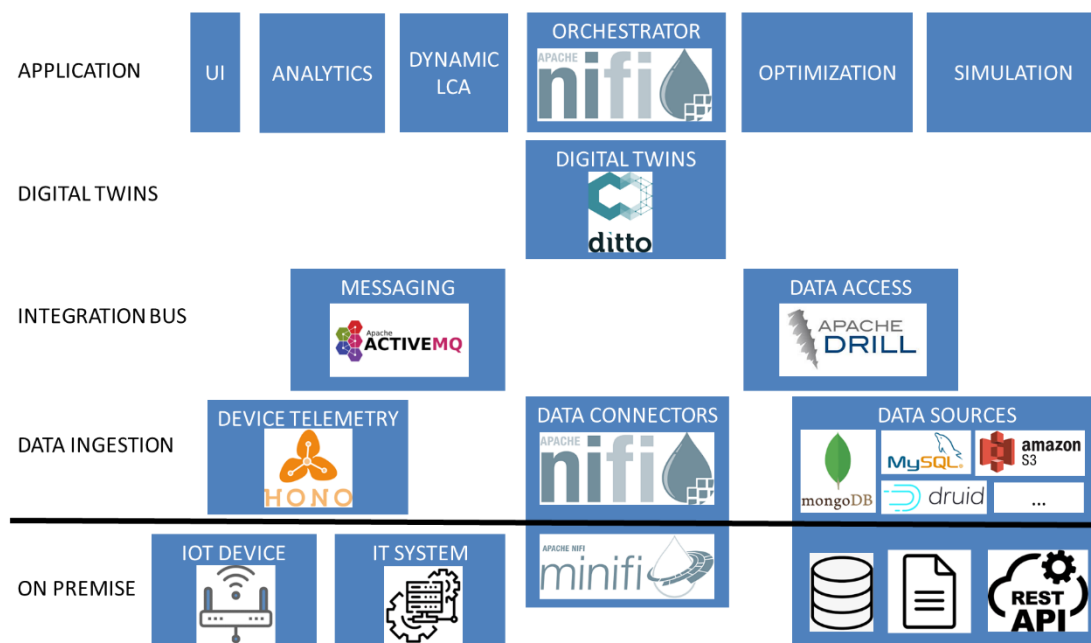


Figure 1 – The database and data-control bus open source technology stack

The AquaSPICE database and data-control bus is comprised of the following layers:

- The **on-premise layer** holds the entities that constitute the primary data producers in the AquaSPICE platform. IoT devices and sensors generate readings that are communicated to the platform either directly to the AquaSPICE ingestion layer or indirectly through an organisation’s IT system. In addition, legacy data sources, e.g., databases, files, or web services, can be attached to the platform and be queried, e.g., for accessing historical data, or monitored for changes.
- The role of the **ingestion layer** is twofold. Firstly, it provides endpoints for device telemetry to be forwarded to the platform. Here the Eclipse Hono solution can be

¹ <https://cordis.europa.eu/project/id/869951>

utilized, offering MQTT, AMQP, or HTTP protocol endpoints. Secondly, it offers a framework for controlling ingestion of data from legacy data sources. Here the Apache NiFi dataflow framework provides a solution for designing and scheduling data import operations, offering components for accessing on premise data sources, transforming data and finally storing them in the appropriate storage solution. In case on premise data processing is needed, the AquaSPICE platform adopts the Apache MiNiFi solution, which acts as a lightweight on-premise agent for the data ingestion layer.

- The **integration bus layer**, following the ingestion of data from the on-premise layer, is responsible for communicating the data to the appropriate components. The AquaSPICE platform adopts the publish/subscribe paradigm and, using Apache ActiveMQ as the message broker, enables platform components to consume data coming from IoT devices, as well as exchange data in an asynchronous manner. Moreover, the integration bus layer provides an interface for accessing the data sources attached to the platform. Here the Apache Drill distributed query engine is leveraged, which enables the SQL-on-anything paradigm. Apache Drill is able to transform SQL queries to a set of underlying data sources, e.g., Mongo DB or even a REST API.
- The **digital twins layer** is in the centre of the platform, since it holds the constantly updated digital representation of the system's state. It acts both as the sink of data coming from the shop floor or from platform components, but also as the source of events representing changes of twin's state. Here, the Eclipse Ditto solution is leveraged making the platform capable of supporting millions of digital twins. Moreover, seamless integration with the underlying data management and ingestion layer is possible since it supports all major communication protocols. Lastly, its API enables management and controlled interaction with them.
- The **application layer** consists of platform services that contribute to the AquaSPICE platform, enhancing the operation of digital twins with their services. The communication of this layer with the digital twins is either direct using their APIs or indirect through the integration bus layer. Lastly, among the applications, the orchestrator has a key role in the AquaSPICE platform, since it drives the AquaSPICE process by connecting analytics, optimization and simulation services to the underlying digital twins. Here, the Apache NiFi dataflow framework will be used, allowing flexible integration between the components and continuous monitoring of its status.

3.2. Connection to the Real-Time Monitoring platform

Although, the WaterCPS platform offers a complete set of supporting services that are vital for the operation of an IoT platform, including device telemetry data ingestion, in AquaSPICE devices and sensors that are deployed in the various case studies will be interfacing with the Real-Time Monitoring Platform.

This is indeed a valuable contribution of the AquaSPICE approach, since having the WaterCPS and RTM platforms as separate but complementary offerings, enables a phased transition towards the modernization of an organisations ICT infrastructure: Firstly, connect and monitor existing devices, sensors and IT systems to the RTM platform. And secondly, deploy the WaterCPS platform in order to exploit the value-added services offered by the analytics, optimization, simulation and dLCA modules.

Regarding integration of the WaterCPS and the RTM platforms, and while having in mind both the database and data-control framework and deliverable D3.1 “RTM Specification and System Architecture” [3], which defines the RTM platform’s architecture, we can identify the following:

From the WaterCPS’ point of view, the RTM platform’s Context Broker has the role of an IT system publishing data to the WaterCPS platform, and the Historical data storage component holds the role of a data source.

From the RTM platform’s point of view, the WaterCPS has the role of an intelligent service subscribing to data and accessing the historical database.

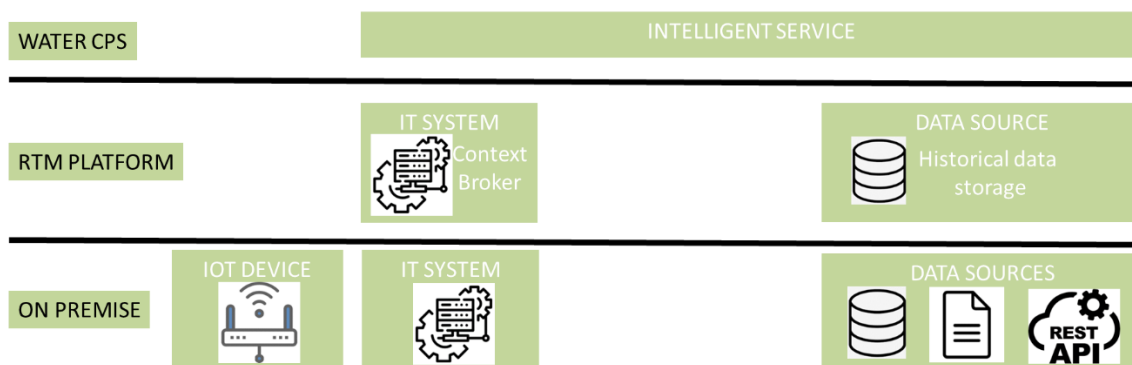


Figure 2 – Connection of the RTM to the WaterCPS platform

The basic mechanism that enables the integration of WaterCPS to the RTM platform’s Context Broker is the pub/sub interface that it offers. The figure below presents the subscription model that it supports. The following aspects can be highlighted:

- Subscriptions can be created for specific entities by using their id, by providing an id pattern or their type. The WaterCPS will leverage these features in order to receive the appropriate data.
- Subscriptions can be granular in terms of which attributes will be monitored. The WaterCPS will be configured according to the case study to monitor a particular set of attributes.
- Subscriptions can have limited number of notifications according to an assigned time interval. The WaterCPS will set these time intervals per case.
- Subscriptions have a designated endpoint where the generated notifications are sent. The WaterCPS will configure this to be the Messaging System’s interface.

```

{
  "id": "URI", # if not given, it will be assigned during subscription process and returned to client
  "type": "Subscription",
  "name": "Name of the subscription",
  "description": "Description of the subscription",
  "entities": [
    {
      "id": "entity id", # Optional, takes precedence over idPattern
      "idPattern": "REGEX", # Optional
      "type": "entity type" # Mandatory
    }
    {
      "id": "entity id", # Optional, takes precedence over idPattern
      "idPattern": "REGEX", # Optional
      "type": "entity type" # Mandatory
    }
  ],
  "watchedAttributes": [ "attr1", "attr2", ..., "attrN" ],
  "timeInterval": Number, # Not Implemented in Alpha Release 1
  "q": "Query Filter",
  "geoQ": {}, # Not Implemented in Alpha Release 1
  "csf": "", # Not Implemented in Alpha Release 1
  "isActive": true/false,
  "notification": {
    "attributes": [ "attr1", "attr2", ..., "attrN" ],
    "format": "keyValues" / "normalized",
    "endpoint": {
      "uri": "URI which conveys the endpoint which will receive the notification",
      "accept": "application/json" / "application/ld+json"
    }
  },
  "status": "ok" / "failed"
},
"expires": "ISO 8601 String",
"throttling": Number, # Minimal period of time in seconds which shall elapse between two consecutive notifications
"temporalQ": # Not Implemented in Alpha Release 1,
"status": "active"/"paused"/"expired" # Read-only - not to be given at creation time
}

```

Figure 3 – RTM platform subscription model

The basic mechanism that enables the integration of WaterCPS to the RTM platform’s Historical data storage component is the REST API that it provides. The figure below presents the query resource and the methods that it offers regarding entities and their attributes, while having the ability to filter the data according to the date they were observed or even applying an aggregation function on them.

queries		^
GET	/v2/entities List of all the entityId	v ↩
GET	/v2/entities/{entityId}/attrs /{attrName}	History of an attribute of a given entity instance. v ↩
GET	/v2/entities/{entityId}/attrs /{attrName}/value	History of an attribute (values only) of a given entity instance. v ↩

Figure 4 – RTM platform data query model

4. Configuring the AquaSPICE Database and Data-Control Bus for the case studies

This chapter presents the workflow and initial setup of the AquaSPICE Database and Data-Control Bus, along with illustrative examples from the project’s case studies demonstrating how each of its layers is configured.

4.1. Methodology

The following figure presents the workflow that is followed in order to setup the database and data-control bus and then configure it for a case study within the AquaSPICE project.

The starting point is the deployment of the platform which involves preparing the server and installing the software libraries that support the technology stack that comprises the database and data-control bus.

Then, follows the connection of the data sources to the platform. This involves configuring the subscription mechanism of the RTM and validating that live data notifications are received.

Once data connectivity is established, developing the digital twins involves modelling the entities of the case study and grouping the data points that are available as data sources to the corresponding entities.

With the digital twins configured, attaching value-added services to them is made possible via the integration layer. Integration handlers implement the APIs and protocols that are exposed by the services. These services are then accessed via the digital twins.

Last step in the workflow is configuring the frontend according to user needs and following an iterative approach, applying any requested changes to the previous steps.



Figure 5 – Database and Data-Control Bus Configuration Workflow

4.2. Deployment

The database and data-control bus consists of a set of open source software, as presented in section 3.1, which altogether offer a powerful technology stack capable of supporting an fully fledged IoT platform such as the WaterCPS. The fusion of these otherwise disparate set of technologies, ranging from databases to web servers and middleware, is made possible with the Docker Engine deployment platform. By adopting the containerization and the micorservices paradigms, the database and data-control bus is easily deployable in host systems that support the docker engine, e.g., Linux or Windows servers.

The following figure presents the containers that comprise the database and data-control bus. The deployment is currently hosted on an Ubuntu 22.04 server with 2 virtual CPUs and 16GB of RAM.

```
aquaspice@Aquaspice-dev:~$ docker ps
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
989776fbca54	quay.io/keycloak/keycloak:20.0.1	"/opt/keycloak/bin/k..."	3 months ago	Up 3 months
ef4a1a493b59	mariaadb:10.3.3	"docker-entrypoint.s..."	3 months ago	Up 3 months
faa5427d3bb8	aquaspice-nifi	". ./scripts/start.sh"	4 months ago	Up 3 months
ff3a9bcfc0f8	aquaspice-httpd	"httpd-foreground"	4 months ago	Up 3 days
784c9969ef44	nginx:1.21-alpine	"docker-entrypoint..."	6 months ago	Up 3 months
af3e1e650286	eclipse/ditto-gateway:2.4.1	"/usr/bin/tini -- sh..."	6 months ago	Up 3 months
cdf979046292	eclipse/ditto-connectivity:2.4.1	"/usr/bin/tini -- sh..."	6 months ago	Up 3 months
356bff21b64d	eclipse/ditto-things-search:2.4.1	"/usr/bin/tini -- sh..."	6 months ago	Up 3 months
fadec03fe563	eclipse/ditto-concierge:2.4.1	"/usr/bin/tini -- sh..."	6 months ago	Up 3 months
a21404de5427	eclipse/ditto-things:2.4.1	"/usr/bin/tini -- sh..."	6 months ago	Up 3 months
03370e9c7768	eclipse/ditto-policies:2.4.1	"/usr/bin/tini -- sh..."	6 months ago	Up 3 months
2b27e9d591d4	mongo:4.2	"docker-entrypoint.s..."	6 months ago	Up 3 months
e2ed0faf1733	swaggerapi/swagger-ui:v4.6.1	"/docker-entrypoint..."	6 months ago	Up 3 months
b7cc44e63653	apache/drill:1.19.0	"/bin/sh -c /opt/dri..."	6 months ago	Up 3 months
22d02f1885bb	mongo	"docker-entrypoint.s..."	6 months ago	Up 3 months
27f271f7b6fa	mongo-express	"tini -- /docker-ent..."	6 months ago	Up 3 months
8a4edb084ef4	activemq	"/docker-run.sh run"	8 months ago	Up 3 months

Figure 6 – Database and data-control bus deployment

4.3. Data ingestion layer

The main goal of the data ingestion layer is the efficient capturing and delivery of IoT data (water quality and purification process measurements) to the integration bus layer. To this end, the AquaSPICE platform distinguishes between IoT devices and legacy data sources and offers tailor-made solutions for handling data ingestion from them. For the former, it leverages the Eclipse Hono solution for capturing device telemetry data, which can be enabled in addition to the Real Time Monitoring platform in case it is needed. For the latter, it employs the Apache NiFi dataflow management solution to design, execute and monitor the data ingestion processes. The sections below present the way that these solutions will be used in AquaSPICE.

4.3.1. Eclipse Hono

The Eclipse Hono solution as presented in [7], is a solution that can be deployed in parallel to the Real Time Monitoring Platform to offer an additional entry point for IoT data. Although, the Eclipse Hono technology suite isn't deployed at the moment, as it can also be seen in the previous figure, in case there is such a demand for a device to directly connect to the database and data data-control bus instead of the Real-Time Monitoring Platform, the device will be registered to the Device Registry of Eclipse Hono and it will be assigned with credentials for publishing data to the platform.

4.3.2. Apache NiFi

The Apache NiFi solution, as presented in more depth in the Annex [7], is a dataflow management solution which enables the AquaSPICE platform to ingest data from legacy data sources, e.g., databases, files and web services. Apache NiFi offers a visual command and control center for designing and deploying dataflows. The following paragraphs present how it is configured for the AquaSPICE project.

The following figure presents the way that the dataflows of each pilot are grouped together. Starting from the bottom, in the first group -called process group in the Apache NiFi terminology- the developer attaches dataflows that are related to data ingestion. In the second process group the developer attaches dataflows that are related to the integration bus and are responsible for message routing from the ingestion layer to the digital twin layer. In the third group, the digital twins process group, the developer attaches dataflows that enrich the digital twins with dynamic behaviour. That is, the group hosts connectors to the value-added services. Lastly, the orchestrator process group holds the dataflows that drive business and operational goals employing the underlying digital twins substrate.

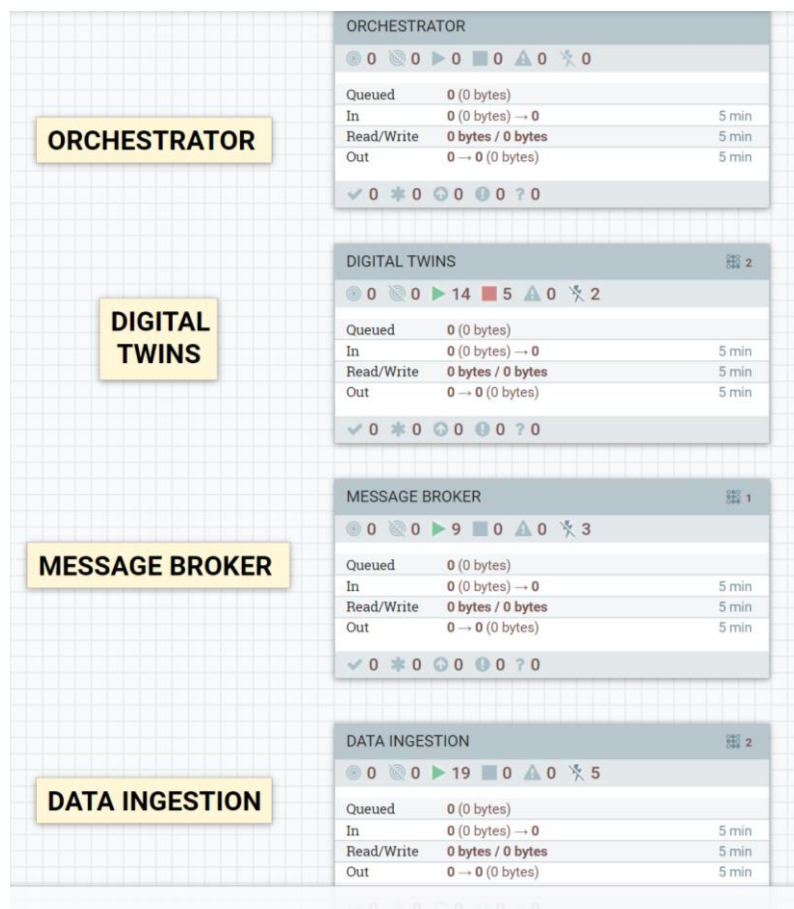


Figure 7 – Apache NiFi – First level configuration for a case study

A fundamental mechanism in the context of the data ingestion layer is the subscription/notification mechanism, as presented in section 3.2, that drives the synchronization of the Real-Time Monitoring platform with the WaterCPS. The figure below presents the subscription that is made for Case Study #3. Line 6 requests from the RTM to notify the WaterCPS of changes in the entities of the RTM that have the type of MeasurementStations. Lines 9-13 configure the way that the WaterCPS will get notified. The notification will be in a key value format and it will be pushed in the appropriate endpoint of the database and data-control bus.

```

1  {
2  .. "id": "AQUASPACE: BASF",
3  .. "type": "Subscription",
4  .. "description": "WaterCPS BASF RTM subscription",
5  .. "entities": [{
6  .. .. "type": "https://cs3.rtm.aquaspice.eurecatprojects.com/schemas/AquaSPICE/MeasurementStation/schema.json"
7  .. .. }
8  .. ],
9  .. "notification": {
10 .. .. "format": "keyValues",
11 .. .. "endpoint": {
12 .. .. .. "uri": "https://io.aquaspice.eu/"
13 .. .. }
14 .. }
15 .. "isActive": true
16 }

```

Figure 8 - RTM-WaterCPS subscription for CS #3

Figure 9 has an example notification, whereas Figure 10 presents the ingestion flow that receives the notifications from the RTM. The dataflow starts with fetching the data mappings between the RTM and the WaterCPS platforms, which are shown in Figure 11. The flow continues to update the appropriate digital twins.

```

1  {
2  .. "id": "urn:ngsi-ld:AquaSpice:measurementStation:KD_01",
3  .. "type": "https://cs3.rtm.aquaspice.eurecatprojects.com/schemas/AquaSPICE/MeasurementStation/schema.json",
4  .. "conductivity": 271.401843,
5  .. "depth": 150.266383,
6  .. "temperature": 233,
7  .. "location": {
8  .. .. "type": "Point",
9  .. .. "coordinates": [
10 .. .. .. 38.337938,
11 .. .. .. 5.995447
12 .. .. ]
13 .. }
14 }

```

Figure 9 – RTM notification example

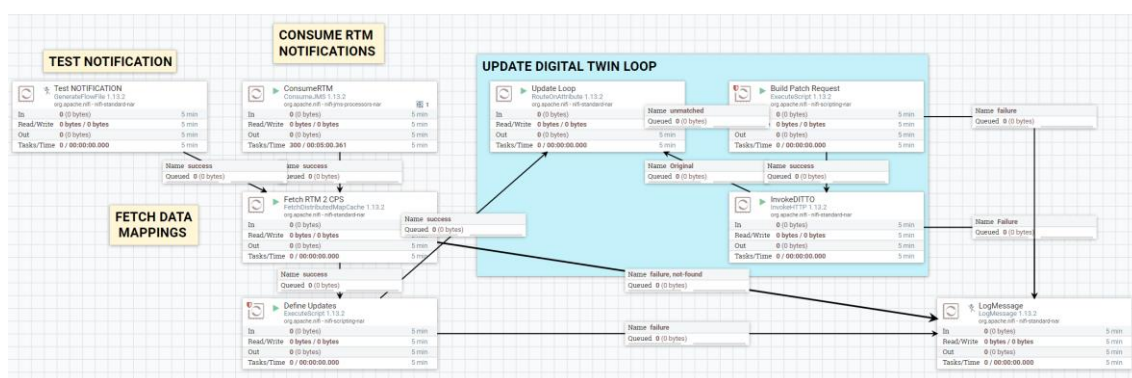


Figure 10 - Ingesting RTM notifications and updating digital twins

```

1 ① {
2 ② "urn:ngsi-ld:AquaSpice:measurementStation:KD_01": {
3 ③   "BASF:KD_01": {
4     "conductivity": ["features/synchronization/properties/status/conductivity/value"],
5     "temperature": ["features/synchronization/properties/status/temperature/value"],
6     "depth": ["features/synchronization/properties/status/depth/value"],
7     "location/coordinates/0": ["attributes/Location/Latitude"],
8     "location/coordinates/1": ["attributes/Location/Longitude"]
9   }
10  },
11 ④ "urn:ngsi-ld:AquaSpice:measurementStation:KD_02": {
12 ⑤   "BASF:KD_02": {
13     "conductivity": ["features/synchronization/properties/status/conductivity/value"],
14     "temperature": ["features/synchronization/properties/status/temperature/value"],
15     "depth": ["features/synchronization/properties/status/depth/value"],
16     "location/coordinates/0": ["attributes/Location/Latitude"],
17     "location/coordinates/1": ["attributes/Location/Longitude"]
18   }
19  },
20 ⑥ "urn:ngsi-ld:AquaSpice:measurementStation:KD_03": {
21 ⑦   "BASF:KD_03": {
22     "conductivity": ["features/synchronization/properties/status/conductivity/value"],
23     "temperature": ["features/synchronization/properties/status/temperature/value"],
24     "depth": ["features/synchronization/properties/status/depth/value"],
25     "location/coordinates/0": ["attributes/Location/Latitude"],
26     "location/coordinates/1": ["attributes/Location/Longitude"]
27   }
28  },
29 ⑧ "urn:ngsi-ld:AquaSpice:measurementStation:KD_04": {
30 ⑨   "BASF:KD_04": {
31     "conductivity": ["features/synchronization/properties/status/conductivity/value"],
32     "temperature": ["features/synchronization/properties/status/temperature/value"],
33     "depth": ["features/synchronization/properties/status/depth/value"],
34     "location/coordinates/0": ["attributes/Location/Latitude"],
35     "location/coordinates/1": ["attributes/Location/Longitude"]
36   }
37  },

```

Figure 11 - RTM-DT platform data mapping

4.4. Integration bus layer

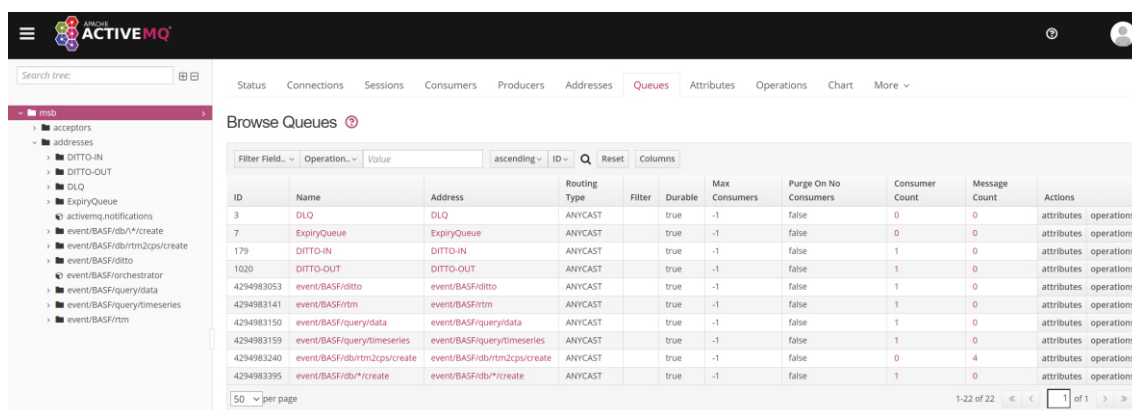
The main goal of the integration bus layer is the efficient routing of water quality and purification process data, as well as AquaSPICE component generated data to the digital twins and application layer. To this end, the AquaSPICE platform distinguishes between messaging and on demand accessing data and offers tailor-made solutions for handling data access in both cases. For the former, it leverages the Apache ActiveMQ solution for routing data to AquaSPICE components. While for the latter, it employs the Apache Drill distributed query engine to provide a unified SQL query interface to a wide variety of data sources. The sections below present the basic concepts and features of both solutions along with the way that they will be used in AquaSPICE.

4.4.1. Apache ActiveMQ

The Apache ActiveMQ message broker, as presented in more depth in the Annex [7], plays a critical role to the platform, since AquaSPICE from the architectural point of view, follows a loosely coupled architecture, where components are decoupled by adopting the publish/subscribe paradigm. Moreover, data ingested in the platform ranging from water quality data (pH, electric conductivity etc.), water purification process data, to data

generated by AquaSPICE components (digital twin state change, analytics, dynamic lifecycle assessment, optimization and simulation results) need to be routed to the interested components in a dynamic way.

The figures below present the message queues that support the operation of WaterCPS and act as a substrate for exchanging messages between layers. For example, the DITTO-IN and DITTO-OUT queues are responsible for the communication with the digital twins layer. Their configuration can be seen in section 4.5.



ID	Name	Address	Routing Type	Filter	Durable	Max Consumers	Purge On No Consumers	Consumer Count	Message Count	Actions
3	DLQ	DLQ	ANYCAST	Filter	true	-1	false	0	0	attributes operations
7	ExpiryQueue	ExpiryQueue	ANYCAST		true	-1	false	0	0	attributes operations
179	DITTO-IN	DITTO-IN	ANYCAST		true	-1	false	1	0	attributes operations
1020	DITTO-OUT	DITTO-OUT	ANYCAST		true	-1	false	1	0	attributes operations
4294983053	event/BASF/ditto	event/BASF/ditto	ANYCAST		true	-1	false	1	0	attributes operations
4294983141	event/BASF/rtm	event/BASF/rtm	ANYCAST		true	-1	false	1	0	attributes operations
4294983150	event/BASF/query/data	event/BASF/query/data	ANYCAST		true	-1	false	1	0	attributes operations
4294983159	event/BASF/query/timeseries	event/BASF/query/timeseries	ANYCAST		true	-1	false	1	0	attributes operations
4294983240	event/BASF/db/rm2cps/create	event/BASF/db/rm2cps/create	ANYCAST		true	-1	false	0	4	attributes operations
4294983395	event/BASF/db/*create	event/BASF/db/*create	ANYCAST		true	-1	false	1	0	attributes operations

Figure 12 - ActiveMQ message queues supporting CS#3

Broker Diagram

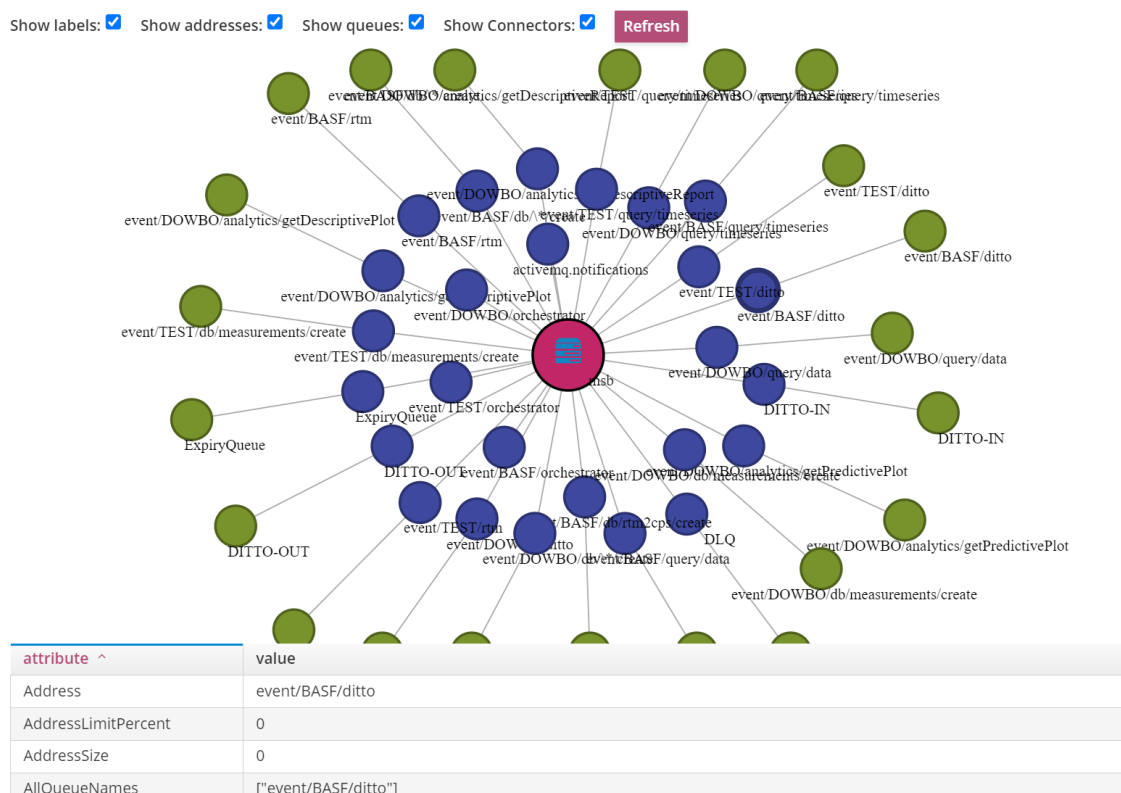
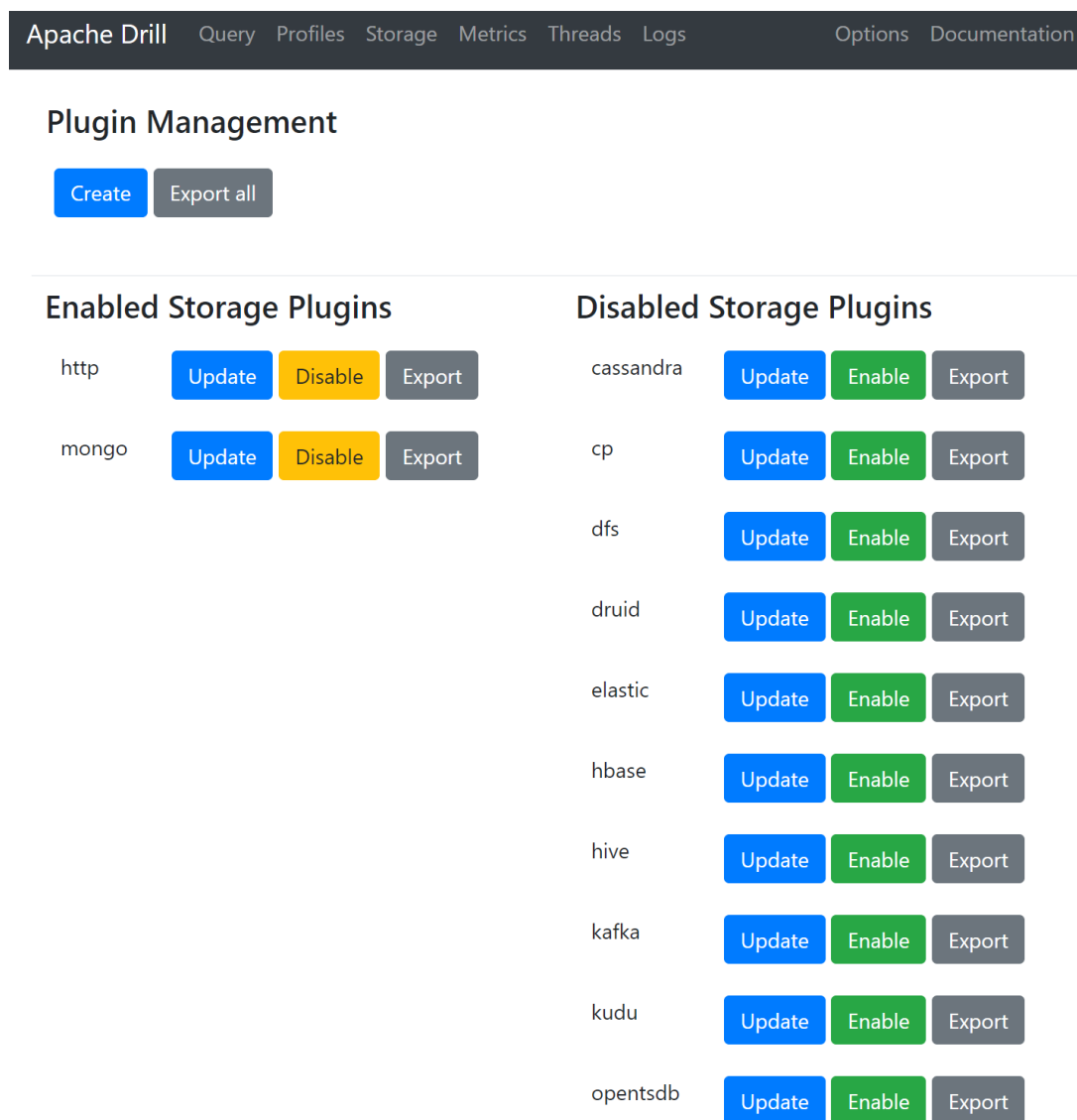


Figure 13 - Apache ActiveMQ message queues interconnection

4.4.2. Apache Drill

The Apache Drill solution, as presented in more depth in the Annex[7], is a core technology of the database and data-control bus, which provides a flexible entry point for data source queries and enables the SQL-on-anything paradigm.

In AquaSPICE, the Mongo DB storage plugin is enabled to persist the data produced in the platform or from the value-added services. However, additional storage plugins (Figure 14) may be enabled if the requirements of the cases require so. Figure 15, for example, presents the dataflow that handles a timeseries query. Upon receipt of the request, the configuration of the digital twin that offers this functionality is fetched. This configuration holds the information needed to construct the query to the underlying data source. The query is forwarded to Apache Drill which abstracts the underlying storage solution, executes the query and returns the results.



The screenshot shows the Apache Drill web interface. At the top, there is a navigation bar with links for Apache Drill, Query, Profiles, Storage, Metrics, Threads, Logs, Options, and Documentation. Below this is the 'Plugin Management' section, which includes 'Create' and 'Export all' buttons. The main content is divided into two columns: 'Enabled Storage Plugins' and 'Disabled Storage Plugins'. Each plugin entry has 'Update', 'Disable', and 'Export' buttons.

Enabled Storage Plugins	Disabled Storage Plugins
<p>http [Update] [Disable] [Export]</p> <p>mongo [Update] [Disable] [Export]</p>	<p>cassandra [Update] [Enable] [Export]</p> <p>cp [Update] [Enable] [Export]</p> <p>dfs [Update] [Enable] [Export]</p> <p>druid [Update] [Enable] [Export]</p> <p>elastic [Update] [Enable] [Export]</p> <p>hbase [Update] [Enable] [Export]</p> <p>hive [Update] [Enable] [Export]</p> <p>kafka [Update] [Enable] [Export]</p> <p>kudu [Update] [Enable] [Export]</p> <p>opentsdb [Update] [Enable] [Export]</p>

Figure 14 - Apache Drill storage plugins

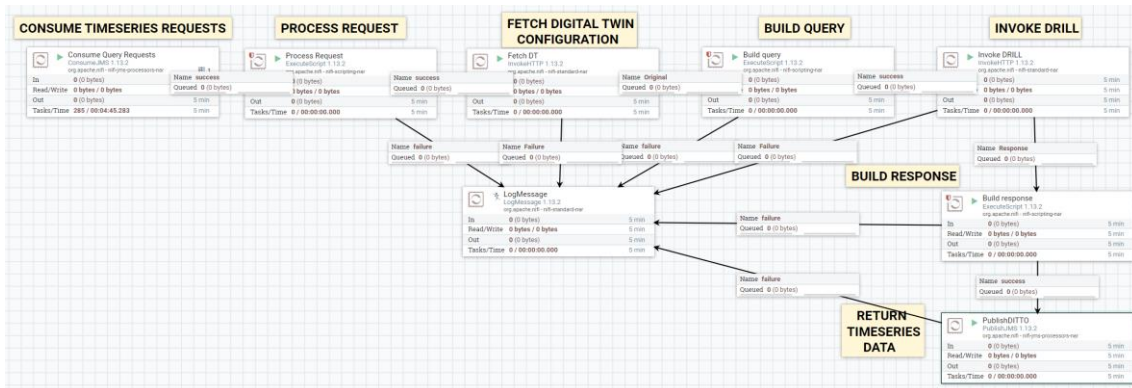


Figure 15 - Integration dataflow handling timeseries queries to the RTM

4.5. Digital Twins layer

In AquaSPICE, all access to the water purification process and the data associated with it is based on the digital twins paradigm. In this regard, the technology of choice for accommodating the specifications presented in D5.1 [5] has been to use Eclipse Ditto, an open-source framework for creating and managing digital twins in the IoT.

4.5.1. Eclipse Ditto

The Eclipse Ditto solution, presented in more depth in the Annex [7], acts as a digital twin middleware, capable of:

- providing a digital representation of real physical devices, offering a consistent view across a variety thereof, e.g., a Water Tank or a Bioreactor;
- acting as broker for communicating with assets, e.g. for actuation purposes;
- facilitating representation of related processes or services, e.g., the WAPERUSE water treatment system of the SOLVAY case study.

At the core of Ditto lies a data model (to be elaborated in the next section) centred around the concept of “Things”, that provide the representations of physical devices. A Ditto Thing is accessible through an API (Figure 16) that allows to interact with the device. This API essentially creates a device-as-a-service for interaction with a digital twin.

Eclipse Ditto™ HTTP API ² OAS3

[/apidoc/openapi/ditto-api-2.yml](#)

JSON-based, REST-like API for Eclipse Ditto

The Eclipse Ditto HTTP API uses response status codes (see [RFC 7231](#)) to indicate whether a specific request has been successfully completed, or not.

The information Ditto provides additionally to the status code (e.g. in API docs, or error codes like, "things:thing.tooLarge") might change without advance notice. These are not be considered as official API, and must therefore not be applied in your applications or tests.

Servers

/api/2 - local Ditto ▾

Authorize 

Things Manage every thing	▾
Features Structure the features of your things	▾
Policies Control access to your things	▾
Things-Search Find every thing	▾
Messages Talk with your things	▾
CloudEvents Process CloudEvents in Ditto	▾
Schemas	▾

Figure 16 - Eclipse Ditto REST API

An essential step for setting up the Eclipse Ditto solution is the creation of a connection to the integration bus layer. Figure 17 presents the sole connection to the integration bus, named DITTO-IO, that consists of a channel for input messages to the digital twins DITTO-IN, and a channel DITTO-OUT for output messages from the digital twins. Lines 22-38 configure an important aspect of the DITTO-OUT channel which is the *target* field. There, the message *header mappings* define that the digital twin id will be appended to all exchanged messages in order to facilitate routing. Additionally, the *topics* field controls which digital twins messages will be forwarded to the integration bus.

```

1  {
2  .. "targetActorSelection": "/system/sharding/connection",
3  .. "headers": {}
4  .. "aggregate": false
5  },
6  .. "piggybackCommand": {
7  .. "type": "connectivity.commands:createConnection",
8  .. "connection": {
9  ..     "id": "DITTO-IO",
10 ..     "name": "DITTO-IO",
11 ..     "connectionType": "amqp-10",
12 ..     "connectionStatus": "open",
13 ..     "uri": "amqp://[REDACTED]",
14 ..     "sources": [
15 ..         {
16 ..             "addresses": [
17 ..                 "DITTO-IN"
18 ..             ],
19 ..             "consumerCount": 1
20 ..         }
21 ..     ],
22 ..     "targets": [
23 ..         {
24 ..             "address": "DITTO-OUT",
25 ..             "topics": [
26 ..                 "_/_/things/twin/events",
27 ..                 "_/_/things/live/messages",
28 ..                 "_/_/things/live/events",
29 ..                 "_/_/things/live/commands"
30 ..             ],
31 ..             "headerMapping": {
32 ..                 "content-type": "{{header:content-type}}",
33 ..                 "reply-to": "{{header:reply-to}}",
34 ..                 "correlation-id": "{{header:correlation-id}}",
35 ..                 "thing-id": "{{thing:id}}"
36 ..             }
37 ..         }
38 ..     ],
39 ..     "clientCount": 1,
40 ..     "failoverEnabled": true,
41 ..     "validateCertificates": true,
42 ..     "processorPoolSize": 5
43 .. }
44 }
45 }

```

Figure 17 - Eclipse Ditto - Connection to the integration Bus

Moving forward to Figure 18, the integration bus listens for the DITTO-OUT channel and performs the message routing using the information contained in the message header, as well as inside the corresponding digital twin.

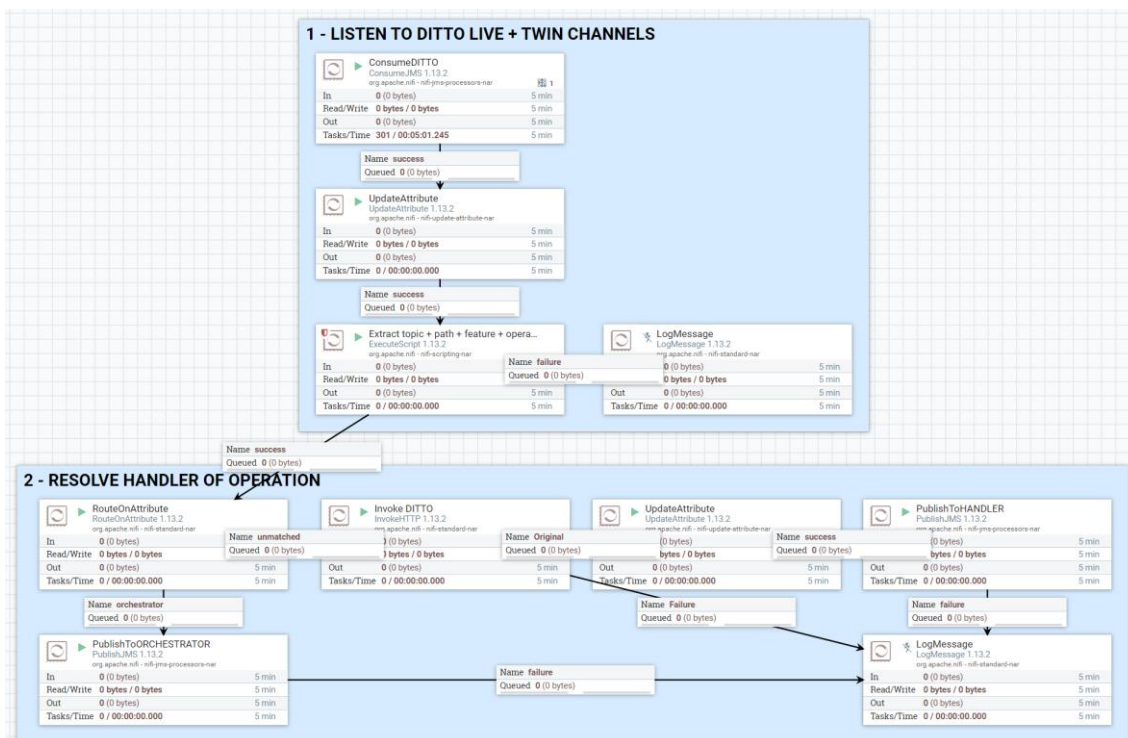


Figure 18 - Eclipse Ditto - Message routing dataflow

4.5.2. Data model

As presented also in more depth in the Annex [7], the definition of the attributes and features of Things, will implement the specifications of ISO/DIS 23247-3 [6]. However, in AquaSPICE the exact content and the specificities of the information elements will depend on the types of things to be addressed on occasion and will be aligned with the descriptions provided for each entity in each case study.

Attributes and features of the digital twins will be grounded to the ontology and data model specified under the activities of Task 1.4 “Data Models, Taxonomy and Ontology for Water Use in the Process Industry” and Task 3.2 “Smart Network with Multi-Source Data Acquisition, Harmonisation and Processing Framework”. The ontology specified in deliverable D1.5, offers semantic grounding to the attributes and features of the digital twins, when viewed as information structures. For example, a temperature attribute will refer to via an accompanying JSON-LD description, the corresponding term in the Smart Data Models ontology (<https://smart-data-models.github.io/data-models/terms.jsonld#/definitions/temperature>). On the other hand, the data model specified in D3.2, is used as a domain model when defining the attributes and features of the digital twins, and acts as the foundation upon which interoperability between the digital twins and the rest of AquaSPICE modules is grounded.

Taking as an example the digital twin that is modelled for the Lake Witznitz in Case Study #1, Figure 19 presents the static information of the Lake that are modelled as attributes.

```

1 {
2   "thingId": "DOWBO:WITZNITZ",
3   "policyId": "DOWBO:WITZNITZ",
4   "attributes": {
5     "identifier": {
6       "assetID": "WITZNITZ",
7       "factlogID": "DOWBO:WITZNITZ"
8     },
9     "characteristics": {
10      "class": "Equipment",
11      "type": "Lake",
12      "description": "This is the digital twin for Lake Witznitz",
13      "name": "Lake Witznitz"
14    },
15    "location": {
16      "address": "Boehlen, Germany",
17      "name": "Lake Witznitz",
18      "latitude": 12.3543,
19      "longitude": 51.1880
20    },
21    "relationships": [
22      {
23        "type": "operates_in",
24        "predicate": "DOWBO:ASIS-PROCESS"
25      },
26      {
27        "type": "controlled_by",
28        "predicate": "DOWBO:BOEHLEN-PLANT"
29      }
30    ]
31  },

```

Figure 19 - Digital Twins Data Model - Static Information

AquaSPICE extends the set of information elements contained in ISO/DIS 23247-3 [6], with the synchronization feature. This structure holds information that is dynamic in nature, but also addresses the requirement of accessing, through a Thing, data not stored within a Digital Twin; this refers mainly to historical data or other information that need to be retrieved on demand from AquaSPICE persistence, on-premise databases or systems or even external sources (e.g., energy cost data over previous days, weeks or months).

Figure 20 presents this structure for the Lake Witznitz case where, the *status* field uses the various measurements that are collected by the sensors and are made available through the RTM. The *operations* field holds the configuration that is needed by the integration bus in order to connect with value-added services, or additional handlers that are exposed via the integration bus. The *alert state* field is filled in cases where issues on the operation of the physical counterpart of the digital twin have been detected, giving additional information on the particular event.

```

32 "features": {
33   "synchronization": {
34     "properties": {
35       "status": {
36         "alert_state":{
37           "phAlert": {
38             "timestamp": 1578261600000,
39             "code": "PH-1",
40             "level": "Critical",
41             "event": {
42               "phValue": 8.1
43             }
44           }
45         }
46       "ph": { "value": "8.1", "timestamp": "2020-01-29T00:00:00+03:00", "unit": "-"},
47       "t": { "value": "17.7", "timestamp": "2020-01-29T00:00:00+03:00", "unit": "C"},
48       "ec": { "value": "496.8", "timestamp": "2020-01-29T00:00:00+03:00", "unit": "µS/cm"},
49       "ks82": { "value": "0", "timestamp": "2020-01-29T00:00:00+03:00", "unit": "mmol/L"},
50       "ks43": { "value": "1.49", "timestamp": "2020-01-29T00:00:00+03:00", "unit": "mmol/L"},
51       "hco3": { "value": "1.49", "timestamp": "2020-01-29T00:00:00+03:00", "unit": "mmol/L"},
52       "co32": { "value": "0", "timestamp": "2020-01-29T00:00:00+03:00", "unit": "mmol/L"},
53       "co2aggr": { "value": "0", "timestamp": "2020-01-29T00:00:00+03:00", "unit": "mg/L"},
54       "tss": { "value": "2", "timestamp": "2020-01-29T00:00:00+03:00", "unit": "mg/L"},
55       "cod": { "value": "17", "timestamp": "2020-01-29T00:00:00+03:00", "unit": "mg/L"},
56       "turbidity": { "value": "1", "timestamp": "2020-01-29T00:00:00+03:00", "unit": "NFU"},
57       "cl": { "value": "55", "timestamp": "2020-01-29T00:00:00+03:00", "unit": "mg/L"},
58       "sulfate": { "value": "82", "timestamp": "2020-01-29T00:00:00+03:00", "unit": "mg/L"},
59       "no3n": { "value": "1.3", "timestamp": "2020-01-29T00:00:00+03:00", "unit": "mg/L"},
60       "no2n": { "value": "0.01", "timestamp": "2020-01-29T00:00:00+03:00", "unit": "mg/L"},
61       "nh4n": { "value": "0.1", "timestamp": "2020-01-29T00:00:00+03:00", "unit": "mg/L"},
62       "ntotanorg": { "value": "1.3", "timestamp": "2020-01-29T00:00:00+03:00", "unit": "mg/L"},
63       "cu": { "value": "0.1", "timestamp": "2020-01-29T00:00:00+03:00", "unit": "mg/L"},
64       "fetot": { "value": "0.06", "timestamp": "2020-01-29T00:00:00+03:00", "unit": "mg/L"},
65       "k": { "value": "6.3", "timestamp": "2020-01-29T00:00:00+03:00", "unit": "mg/L"},
66       "na": { "value": "37", "timestamp": "2020-01-29T00:00:00+03:00", "unit": "mg/L"},
67       "camg": { "value": "1.9", "timestamp": "2020-01-29T00:00:00+03:00", "unit": "mmol/L"},
68       "ca": { "value": "1.4", "timestamp": "2020-01-29T00:00:00+03:00", "unit": "mmol/L"},
69       "znfe": { "value": "0.01", "timestamp": "2020-01-29T00:00:00+03:00", "unit": "mg/L"},
70       "siozn": { "value": "2.07", "timestamp": "2020-01-29T00:00:00+03:00", "unit": "mg/L"},
71       "po4ptot": { "value": "0.21", "timestamp": "2020-01-29T00:00:00+03:00", "unit": "mg/L"},
72       "orthopo4p": { "value": "0.18", "timestamp": "2020-01-29T00:00:00+03:00", "unit": "mg/L"},
73       "fedissolved": { "value": "0.01", "timestamp": "2020-01-29T00:00:00+03:00", "unit": "mg/L"},
74       "zndissolved": { "value": "0.01", "timestamp": "2020-01-29T00:00:00+03:00", "unit": "mg/L"},
75       "tc": { "value": "24", "timestamp": "2020-01-29T00:00:00+03:00", "unit": "mg/L"},
76       "tic": { "value": "17", "timestamp": "2020-01-29T00:00:00+03:00", "unit": "mg/L"},
77       "toc": { "value": "6.5", "timestamp": "2020-01-29T00:00:00+03:00", "unit": "mg/L"},
78       "hg": { "value": "0", "timestamp": "2020-01-29T00:00:00+03:00", "unit": "mg/L"},
79       "arsen": { "value": "0", "timestamp": "2020-01-29T00:00:00+03:00", "unit": "mg/L"},
80       "cd": { "value": "0", "timestamp": "2020-01-29T00:00:00+03:00", "unit": "mg/L"},
81       "chr": { "value": "0", "timestamp": "2020-01-29T00:00:00+03:00", "unit": "mmol/L"},
82       "pb": { "value": "0", "timestamp": "2020-01-29T00:00:00+03:00", "unit": "mmol/L"},
83       "bacterialcount": { "value": "3", "timestamp": "2020-01-29T00:00:00+03:00", "unit": "NFU"}
84     },
85     "operations":{
86       "getTimeseries": {
87         "handler": "query/timeseries",
88         "queryStr": "select `timestamp`, value from mongo.dowbo.measurements where thing='DOWBO:WI'
89       }
90     }
91   }

```

Figure 20 - Digital Twins Data Model – Synchronization Feature

Lastly, the following figure presents the way that a value-added service is represented in a digital twin. The value-added service is modelled as a feature of the digital twin, and it has information on the *configuration* of the service, the available *operations* as well as the *status* or latest results of the service.

```

88  "lca": {
89    "properties": {
90      "configuration": {},
91      "operations": {
92        "getTimeseries": {
93          "handler": "query/timeseries",
94          "queryStr": "select `timestamp`, value from mongo.dowbo.measurements where thing='DOWBO:ASIS-PROCESS' and key='$id' and ct
95        },
96        "listLCAs": {
97          "handler": "query/data",
98          "queryStr": "select id, request_timestamp, response_timestamp, status, initiator, operation from mongo.dowbo.Lcas order by
99        },
100       "getLCAById": {
101         "handler": "query/data",
102         "queryStr": "select id, status, request_timestamp, result, response_timestamp from mongo.dowbo.Lcas where id = '$id'"
103       }
104     },
105     "status": {
106       "electricity": { "value": "39.43", "timestamp": "2020-01-29T00:00:00+03:00", "unit": "%"},
107       "lime": { "value": "5.53", "timestamp": "2020-01-29T00:00:00+03:00", "unit": "%"},
108       "ironchloride": { "value": "55.04", "timestamp": "2020-01-29T00:00:00+03:00", "unit": "%"},
109       "carbon": { "value": "27.05", "timestamp": "2020-01-29T00:00:00+03:00", "unit": "kgCO2,eq/hr"},
110       "eutrophication": { "value": "0.09", "timestamp": "2020-01-29T00:00:00+03:00", "unit": "kgPO4,eq/hr"}
111     }
112   }
113 }

```

Figure 21 - Digital Twins Data Model - Value-Added Service feature

5. Conclusions

The AquaSPICE database and data-control bus framework as described in this deliverable provides the foundations for data and services integration within the WaterCPS platform. The framework, a technology stack originally compiled under the context of the FACTLOG project [7], takes into account and evolves the main data communication and integration principles set forth within the deliverable D5.1 “WaterCPS Services Specifications, Functional Model and System Design” [5]. In this direction, it provides, on the one hand, the technical specifications meant to implement the Integration Bus functionalities in terms of data ingestion and service integration, and, on the other, some insights on how the digital twins platform deployed for AquaSPICE will make collected data accessible to all other interested applications and modules.

The deliverable has focused on three layers of the framework. First, on the data ingestion layer, which is the main data entry point of the WaterCPS and the place where the WaterCPS interfaces with the Real-Time Monitoring platform. Second, on the integration bus layer, which focuses on implementing the identified interaction patterns, and setting the ground for the interoperation of AquaSPICE services. Third, to describe the Digital Twins layer, which implements the digital twins paradigm applied in the case studies of AquaSPICE.

Lastly, certain aspects of the framework will be further elaborated in D5.4/D5.5/D5.6 “WaterCPS Digital Platform prototype (1st / 2nd / Final), which will constitute significant platform milestones, and, as such, will validate the application of the proposed database and data-control bus mechanisms, as well as present any potential shortcomings, guidelines and best practices for their configuration.

6. References

- [1] AquaSPICE Deliverable D1.3, “Use Cases Definition with Baseline Assessment”, November 2021.
- [2] AquaSPICE Deliverable D1.5, “Data Models, Taxonomy and Ontology Industrial Water Use”, March 2022.
- [3] AquaSPICE Deliverable D3.1, “RTM Specification and System Architecture”, November 2021.
- [4] AquaSPICE Deliverable D3.2, “Data Management and Harmonization Plan”, March 2022.
- [5] AquaSPICE Deliverable D5.1, “WaterCPS Services Specifications, Functional Model and System Design”, January 2022.
- [6] ISO 23247-3:2021, “Automation systems and integration — Digital twin framework for manufacturing — Part 3: Digital representation of manufacturing elements”, October 2021.
- [7] FACTLOG Deliverable D6.2, “Data Collection Framework (Final Version)”, January 2022

7. Annex: Open source technology stack overview

The Annex presents detailed technical descriptions of the open source technology stack, with technologies developed and maintained by the Eclipse Foundation and the Apache Software Foundation, that comprises the database and data-control bus. This technology stack is shared with the FACTLOG project, as it is reported in Deliverable D6.2 of the FACTLOG project [7]. The descriptions are added here for completeness, along with AquaSPICE specific adaptations and configurations.

7.1. Eclipse Hono

Given the communication protocol landscape in the domain of IoT, the need for a framework where the protocol used to communicate with a device would be hidden from IoT applications came forward. The Eclipse Hono solution covers this need by providing remote service interfaces for connecting IoT devices to applications enabling interactions with them in a uniform way regardless of the device communication protocol. The figure below illustrates the fundamental concept behind the Eclipse Hono solution that allows for a uniform interface for IoT applications to access, command and control IoT devices, independently from the communication protocol that they use.

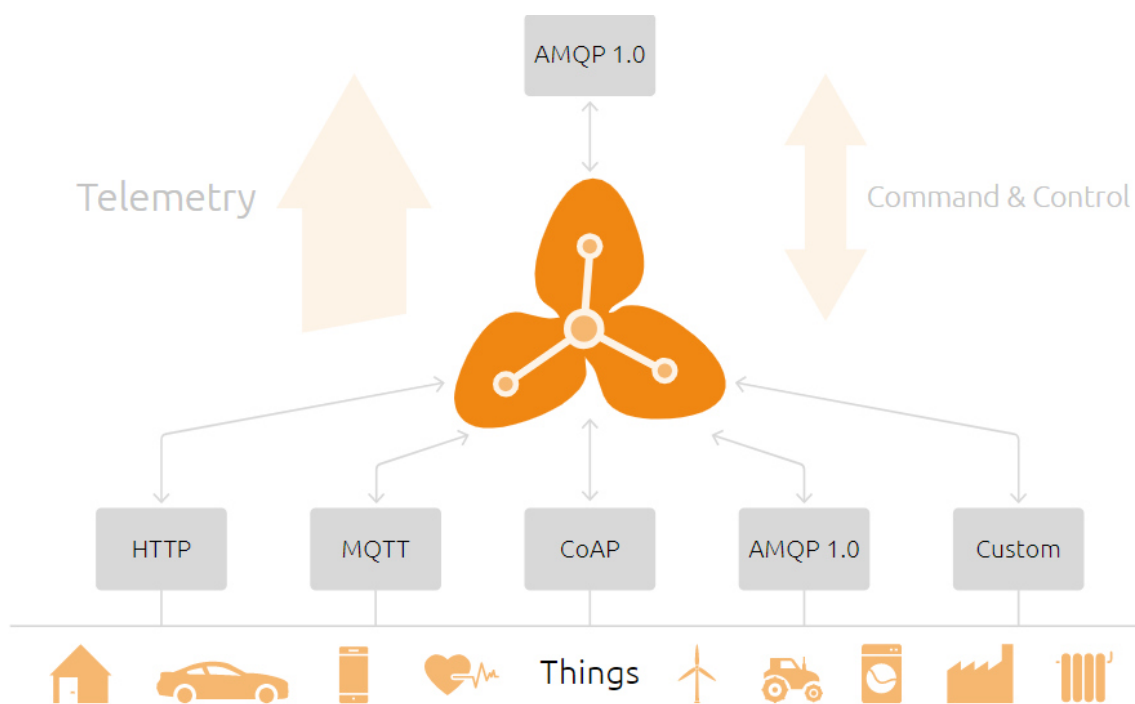


Figure 22 – Eclipse Hono overview

Eclipse Hono is designed for connecting large numbers of IoT devices. It follows the micro services architecture paradigm and uses reactive programming to achieve horizontal scalability. It supports common IoT protocols like HTTP, MQTT, AMQP and CoAP and provides APIs for Telemetry messages to report sensor readings whereas applications can use Command & Control to invoke operations on devices. Moreover, it supports common authentication mechanisms like username/password and X.509 client certificates to

verify a device’s identity and uses transport layer security when communicating with devices.

The figure below presents the components of Eclipse Hono architecture. Starting from the left, it deploys protocol adapters that offer endpoints for devices to communicate their readings. The deployment of the adapters is managed from the device registry, which holds the configuration for each device and employs the authentication service to control the access to these endpoints. After a message has been successfully received, it is forwarded to the data management layer through an AMQP messaging interface. In order to monitor the infrastructure, Eclipse Hono deploys additional services like Grafana for visualizing and Prometheus for collecting the device message logs.

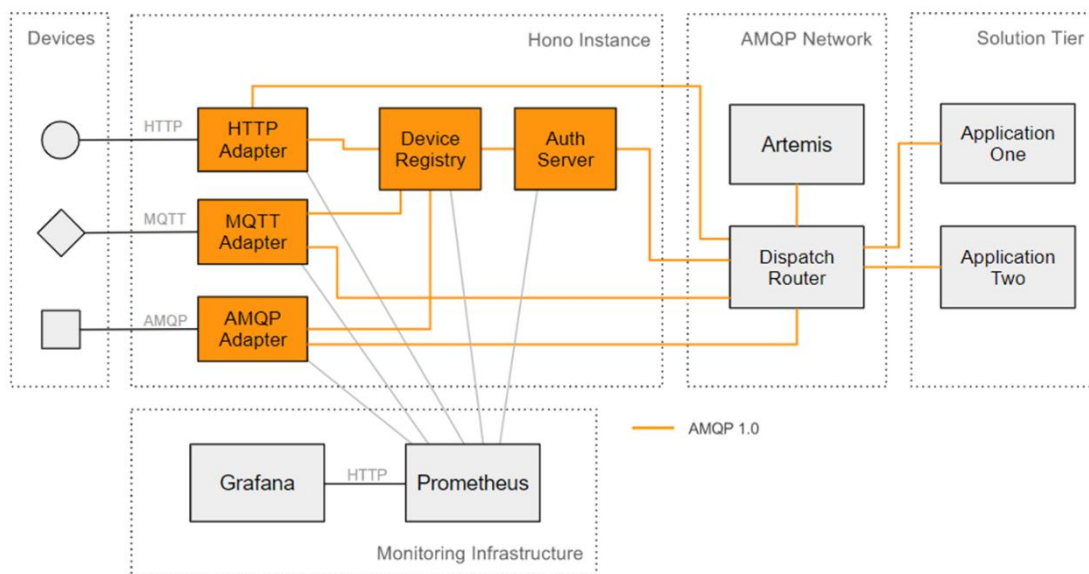


Figure 23 – Eclipse Hono architecture

In AquaSPICE each device that needs to be directly connected to the platform instead of the Real-Time Monitoring Platform will be registered to the Device Registry, and will be assigned with credentials for publishing data to the platform. In case direct communication with the platform isn’t an option or an existing IT system handles communication with the devices, the Eclipse Hono Device Telemetry API will be used to push data to the platform.

7.2. Apache NiFi

The ingestion of data from legacy data sources, e.g., databases, files and web services, poses to the AquaSPICE platform different requirements in relation to the previous case. Indeed, the communication protocol plethora still remains, but here it is assumed that the model of interaction follows the pull approach. For example, daily synchronization of water quality measurements might be required in order to compute an optimal configuration for the water purification process.

In order to achieve scenarios like the one described above, the AquaSPICE platform adopts the Apache NiFi dataflow management solution providing a user-friendly way to connect to the various data sources. Apache NiFi offers a visual command and control center for designing and deploying dataflows.

The “**dataflow**” term stems from the domain of Flow Based Programming, and captures essentially an executable definition of a data exchange scenario between information systems, modeling their interactions as processors exchanging flow files via connections.

- **Flow files** represent the data objects moving through the system. NiFi keeps track of their attributes in key/value pairs along with their associated content.
- **Processors** perform data routing, transformation, or mediation between systems, using the attributes and content of a given flow file.
- **Connections** provide the linkage between processors, as edges of the dataflow directed graph. They act as queues and allow various processors to interact with each other.

Execution of dataflows takes place in the flow controller of Apache NiFi, which acts as an orchestrator, maintaining the knowledge of how dataflows connect, and facilitating the exchange of flow files between processors.

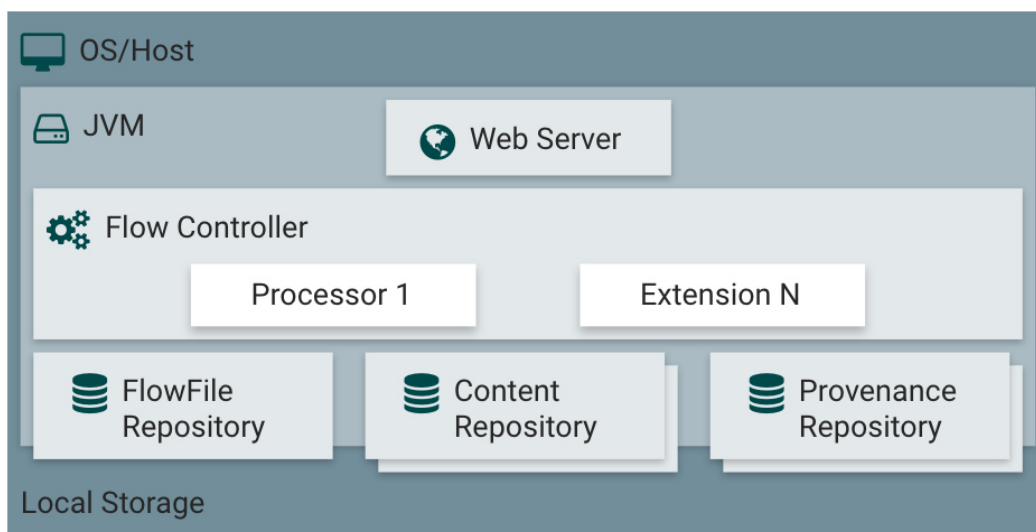


Figure 24 – Apache NiFi architecture

As shown in the figure above with the architecture of Apache NiFi, the Web Server provides the web-based command and control interface. Flow Controller is the dataflows orchestrator between Processors. NiFi provides a rich set of build-in processors that cover the common cases. Extensions allows developers to add functionalities to the application in order to meet their needs. FlowFile Repository is used for storing the state of what is known about a given flow file that is active in a dataflow while the Content Repository keeps track of the actual content bytes of data files. Provenance events are stored in the Provenance Repository, in a way that each FlowFile’s history can be retrieved. It is noted

that for all these three repositories default implementations are provided by the NiFi tool, but the NiFi pluggable architecture allows custom implementations.

Apache NiFi provides a rich set of processors that can be used for:

- data ingestion into the NiFi data flow (e.g. GetFile, GetHTTP, GetFTP, GetKAFKA),
- routing and mediation of data flows (e.g. RouteOnAttribute, RouteOnContent, ControlRate, RouteText), database access (e.g. ExecuteSQL, PutSQL, PutDatabaseRecord, ListDatabaseTables),
- extracting, analyzing or changing flowfile attributes processing in the NiFi data flow (e.g. UpdateAttribute, EvaluateJSONPath, ExtractText, AttributesToJSON),
- running processes or commands in any operating system (e.g. ExecuteScript, ExecuteProcess, ExecuteGroovyScript, ExecuteStreamCommand),
- transforming content (e.g. ReplaceText, JoltTransformJSON),
- sending data (e.g. PutEmail, PutKafka, PutSFTP, PutFile, PutFTP),
- splitting and merging the content present in a flowfile (e.g. SplitText, SplitJson, SplitXml, MergeContent, SplitContent),
- processing of HTTP and HTTPS calls (e.g. InvokeHTTP, PostHTTP, ListenHTTP)
- interacting with Amazon web services system (e.g. GetSQS, PutSNS, PutS3Object, FetchS3Object).

An example dataflow for fetching a measurement entry based on the device identifier (deviceId) is shown in the figures below. The presented dataflow contains both data transformation and communication to an existing database. For simplicity, it is assumed that the dataflow gets as input a JSON object as a String with only the device identifier (for example, “{‘sourceDevId’:1}”) and produces a JSON object containing all database table entry columns that are stored in a local file.

The GenerateFlowFile NiFi processor is used for storing the JSON String to a NiFi FlowFile. Then, the EvaluateJSONPath processor is used for extracting the “sourceDevId” field from the JSON String and store it to the “deviceId” NiFi attribute (cf. Figure 26). The ExecuteSQL processor performs the actual communication to the database utilizing the “deviceId” attribute (cf. Figure 27). The SplitAvro process is used for splitting the database query result into smaller files, that are passed to the ConvertAvroToJSON in order to be converted into a JSON representation. Finally, the JSON data are stored into a local file by the PutFile processor.

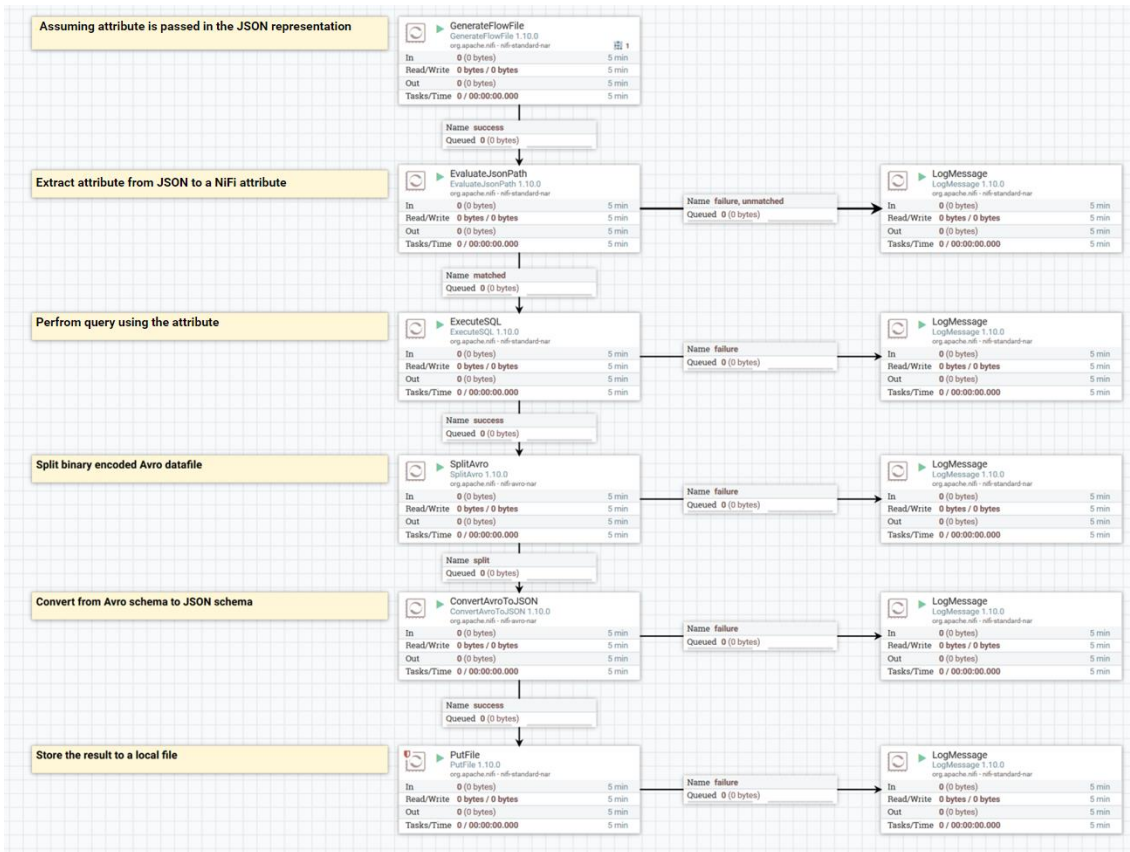


Figure 25 – Data connector simple example

Configure Processor

Stopped

SETTINGS | SCHEDULING | **PROPERTIES** | COMMENTS

Required field +

Property	Value
Destination	flowfile-attribute
Return Type	auto-detect
Path Not Found Behavior	ignore
Null Value Representation	empty string
deviceld	\$.sourceDevID 🗑

Figure 26 – EvaluateJSONPath NiFi processor properties

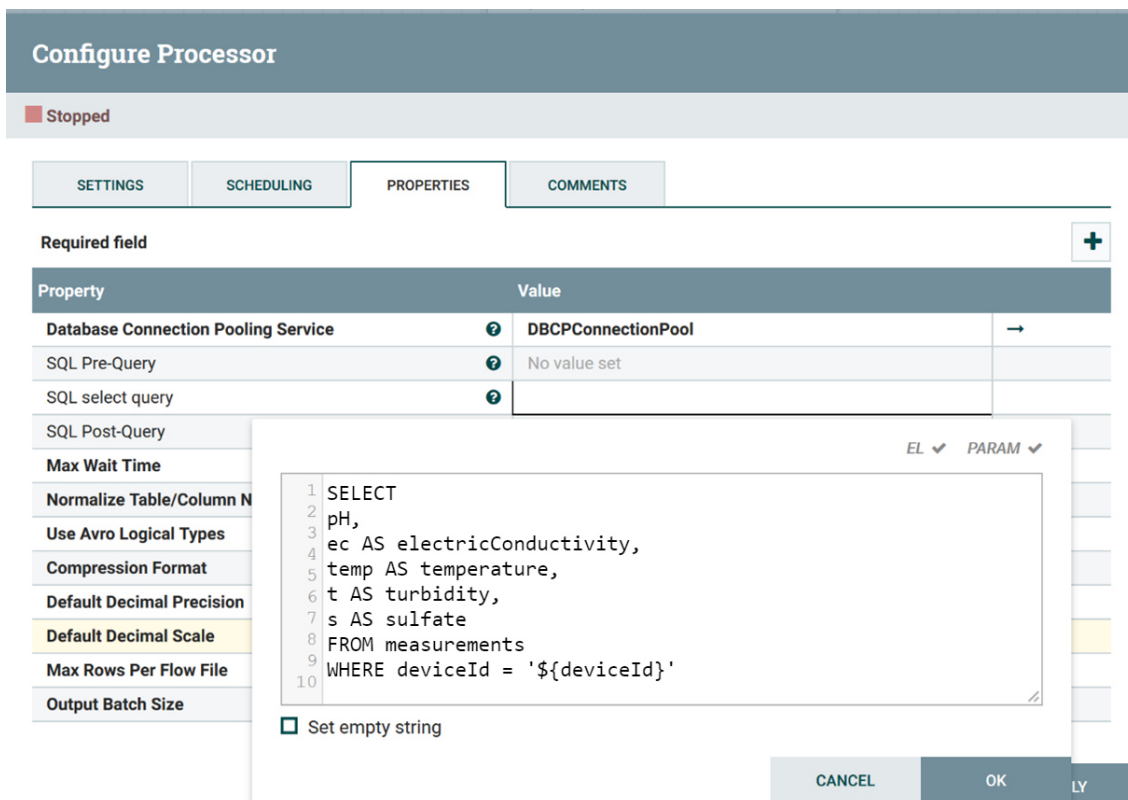


Figure 27 – ExecuteSQL NiFi processor properties

7.3. Apache ActiveMQ

The role of the Apache ActiveMQ message broker is critical to the platform, since AquaSPICE, from the architectural point of view, follows a loosely coupled architecture, where components are decoupled by adopting the publish/subscribe paradigm. Moreover, data ingested in the platform ranging from water quality data (pH, electric conductivity etc.), water purification process data, to data generated by AquaSPICE components (digital twin state change, analytics, dynamic lifecycle assessment, optimization and simulation results) need to be routed to the interested components in a dynamic way.

Apache ActiveMQ supports message routing enterprise integration patterns by providing both asynchronous and point-to-point message exchange between system components; it further supports streaming, enabling on-the-fly and real-time processing of data as it arrives. Apache ActiveMQ supports all major standard protocols so application layer services can be developed in a broad range of programming languages. In that respect, it can handle any messaging use-case, from routing IoT device messaging using the MQTT protocol, to delivering messages to distributed services using the AMQP protocol.

In order to support AquaSPICE use cases, data will be routed, as shown in the figure below, to pre-defined topics and queues, so that data ranging from Digital Twins lifecycle events to breakdown analytics, orders, production schedule, predictions, variation detections, device telemetry, simulation or optimization results reach the appropriate destinations.

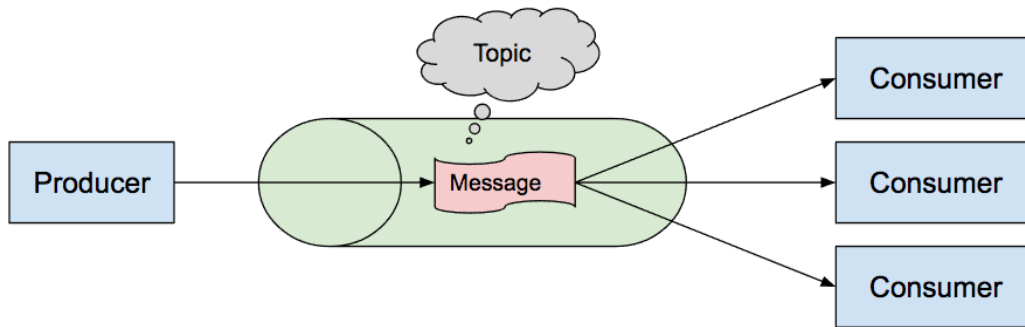


Figure 28 – Apache ActiveMQ - Publish/Subscribe

7.4. Apache Drill

A core technology used in the data management layer is the Apache Drill solution. Apache Drill is a distributed query engine which leverages the SQL query language, offering abstractions over a variety of data source technologies, relational, non-relational, files or even web services. Apache Drill provides a flexible entry point for data source queries and enables the SQL-on-anything paradigm.

Apache Drill supports a variety of NoSQL databases and file systems, including HBase, MongoDB, HDFS, Amazon S3, Azure Blob Storage, Google Cloud Storage local files and HTTP web services. A single query can join data from multiple data sources. For example, it can join machine records in MongoDB with machine maintenance schedules stored as files in a directory.

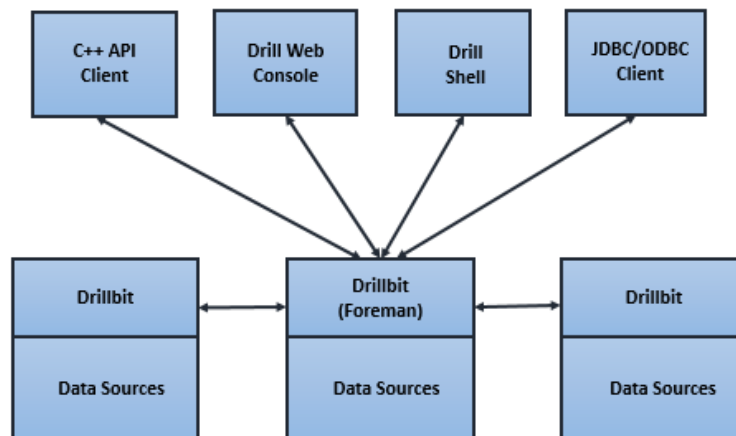


Figure 29 – Apache Drill interfaces

Apache Drill offers APIs for accessing the SQL interface and introduces the concept of Drillbits, standard adapters to the underlying data source technology. Internally, a Drillbit, as shown in the figure below, accepts from the remote procedure call endpoint a query, which is then forwarded to the SQL parser. The query is transformed using the optimizer and finally is executed using the appropriate storage engine adapter.

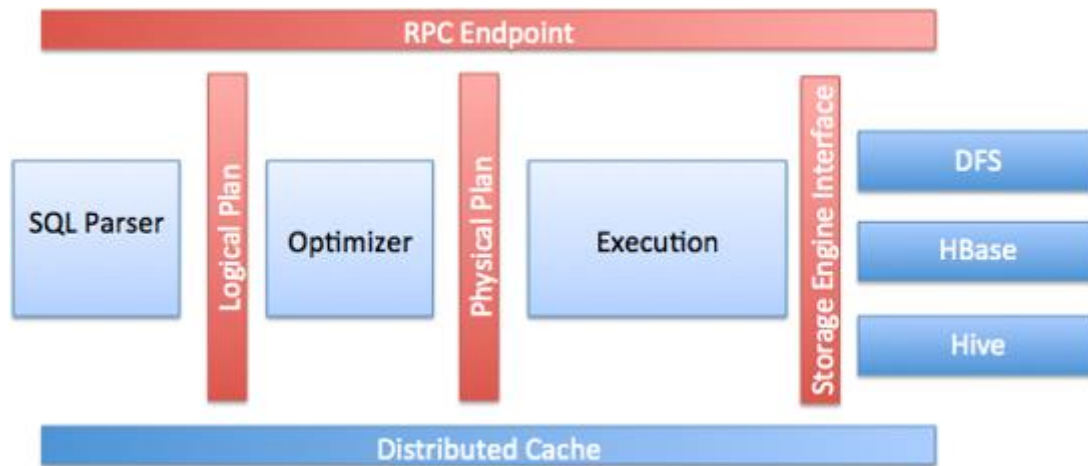


Figure 30 – Apache Drill - Drillbit architecture

Configuration of the data sources can take place via the user interface, REST API or using files. Although the configuration of each data source differs, the common configuration parameters are shown in the picture below:

```
{
  "type": "file",           _____ Type of storage plugin: file, hbase, hive, or mongo
  "enabled": true,         _____ State of the storage plugin
  "connection": "file:///", _____ Connection string to the data source, such as the distributed or embedded file system
  "workspaces": {
    "root": {
      "location": "/",     _____ Path to workspace
      "writable": false,   _____ Allows or disallows creating a table or a view in the workspace
      "defaultInputFormat": null _____ Default format Drill reads regardless of extension
    },
    . . .
  },
  "formats": {
    "psv": {               _____ Data format name
      "type": "text",     _____ Type of formatting
      "extensions": [
        "tbl"             _____ Extension of files Drill can read for this type of format
      ],
      "delimiter": "|"    _____ Character that Drill recognizes as the delimiter for the text format
    },
    . . .
  }
}
```

Figure 31 – Apache Drill - Drillbit configuration

7.5. Eclipse Ditto

At the core of Ditto lies a data model centred around the concept of “Things”, that provide the representations of physical devices. The Ditto Thing is accessible through an API that allows to interact with the device. This API essentially creates a device-as-a-service for interaction with a digital twin. Ditto services support interaction with the data model basically through the features presented in what follows.

Functional view

Persistence of device state

In principle, devices are not always connected to the network, however applications always need to be able to access their data. To address this, Ditto takes over saving the most recent values of a device in a MongoDB database, allowing digital twins to query the last reported value of a device; this way, at any given point in time, Ditto universally represents the current state of the digital twin. In this case Ditto is used itself for persistence, enforcing access control (see below), processing commands and emitting events. In other words, this channel connects to the digital representation of a Thing; this Thing is managed with Ditto and its state and properties can be read and updated.

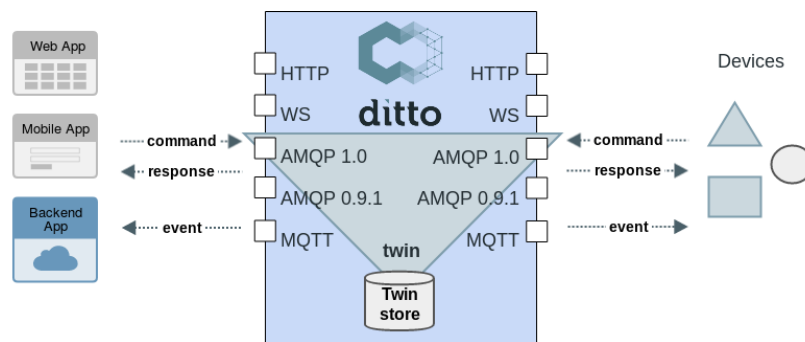


Figure 32 – Eclipse Ditto – Twin channel

However, historical values are not persisted in Ditto. In order to cover the need to access historical data, AquaSPICE makes use of a connection from Ditto to the messaging system in place, which may get all twin change events and puts the historical data to a data store suitable for persisting and querying such data, e.g., into a timeseries database such as the one that is offered by the Real-Time Monitoring platform.

Live channel

In addition to the persistent mode, Ditto has a “live” channel which lets an application communicate directly with a device. Using live channel, Ditto acts as a router forwarding requests via the device connectivity layer, implemented here by Hono, to the actual devices. This channel can also be used to invoke operations (e.g., “turn on/off”) on the device and accept a response back from a device; in this case endpoints process commands and emit events. Ditto live channel also checks the authorization policies for a device to ensure only authorized clients have access to the device information. From there on, the handling and execution of a received command/message by a device (or a gateway which connects the device) is performed by Hono adapters and the Real-Time Monitoring platform.

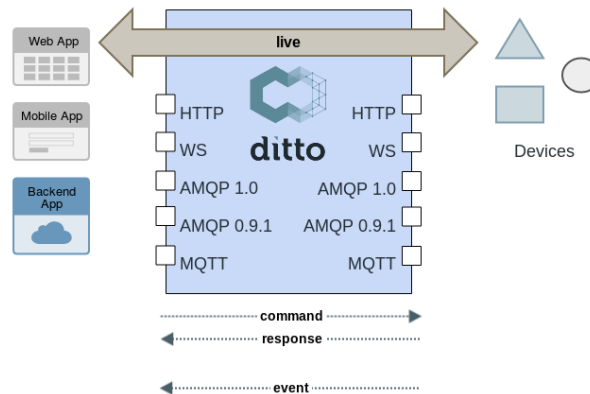


Figure 33 – Eclipse Ditto – Live Channel

Authorisation

Ditto can restrict access to the APIs through a built-in authorisation mechanism based on predefined authorisation policies, which can be specified as fine-grained as necessary for the respective use case. Ditto authorisation services protects the privacy and integrity of the device data. Only predefined authorised clients are granted read/write access to individual elements of a Ditto Thing. Clients are authenticated in Ditto using the OAuth 2.0 and OpenID Connect standard.

Search

A number of fundamental search capabilities across a large number of devices is also supported. In particular, Ditto utilizes a subset of RQL as language for specifying queries against arbitrary data, while covering aspects like authorisation, field projection over the results and indexing.

Search queries may include generating a list of all current twins or searching against reported data, search for twins above a certain data threshold, etc. Search is also supported to query against device meta information, in order to, e.g., list all of twins that represent temperature sensors. Search services in AquaSPICE can be used for a variety of purposes, according to requirements posed by the rest of the modules as well as the front end, in order, for example, to create a dashboard to show real-time water quality data.

Payload normalization

As aforementioned, Ditto provides structured APIs of things and is device and domain agnostic. At the same time, the way that data are transmitted and formatted according to each individual production environment and application domain may vary. In order to bridge the gap, Ditto allows for the mapping of different device data into a consistent lightweight JSON model, enabling the transformation of both incoming and outgoing data and, hence, resulting in a consistent interface for a heterogeneous set of devices.

Interfaces

Ditto basically provides two ways to interact with: a) a REST-like HTTP API with a sophisticated resource layout that allows to create, read, update and delete Things and the Thing's data; b) a JSON-based WebSocket API implementing the Ditto Protocol. The two ways are almost equally powerful and allow the same operations to work with the Thing's data, send messages to Things and receive messages from Things. As a rule of thumb, the lightweight REST-like HTTP API will be used:

- on less powerful devices lacking a Java runtime or supporting other (scripting) languages like JavaScript, Python, C/C++,
- for developing Web-based user interfaces.

On the other hand, the WebSocket API will be chosen for:

- gathering data streams from devices or massive data from message brokers,
- real-time device monitoring,
- event-driven Web applications,
- full duplex communication scenarios, etc.

Ditto also offers a Connectivity API for the management of client connections to remote systems, like, for instance, external messaging services, and the exchange of Ditto Protocol messages with those. More specifically, it supports the following connection types: AMQP 0.9.1, AMQP 1.0, MQTT 3.1.1, MQTT 5, HTTP 1.1, Kafka 2.x. In AquaSPICE this feature will be used in order to integrate the Ditto instance with data coming from the data ingestion and integration bus layers.

On the basis of the above APIs, the interactions that will be supported by Ditto in AquaSPICE could be grouped as follows:

- **Management:** Digital twins can be added, updated or deleted, either manually or automatically as triggered by events coming from the physical infrastructure itself, as a physical asset and its physical relations with other assets (such as sensor physical fixation position, etc.) can be changed very often during its lifecycle; in this sense synchronisation of the status of each digital twin with the status of the corresponding ME is facilitated. Lookup functionality for digital twins is also offered to external entities, leveraging Ditto search features mentioned above. Additionally, the platform supports capabilities related to its overall operation and management, including providing administration functionalities to the end-user. The latter may concern the management of connections to external systems, of policies and any other configuration.
- **Data acquisition:** Operation data can be transferred and recorded by the digital twin. Such data may concern raw data captured with monitoring and sensing devices and transmitted through the message brokers, or pre-processed data (e.g., after filtering, aggregation, other cleansing functions). This data can be stored and then queried by the digital twin. A digital twin can also establish that it needs to be notified when the value changes, as supported through the connected message brokers. Above

functionality also plays a big part in the synchronisation of the digital twins with their physical counterparts in terms of real-time continuous updates.

- **Data access:** The AquaSPICE business domain, i.e., any interested applications (e.g., optimization of water purification process), reporting, dashboards, etc., acquire access to the various digital twins aspects, functionalities and stored data generated throughout their lifecycle. This can take place either through the twin channel, when this concerns current state data locally stored in Ditto, or through the live channel, covering, for instance, cases where the corresponding information is kept on premise and accessed on demand, through commands/messages towards an actual device/system. Associated information is made available by the digital twins as events.
- **Feedback:** Commands, on the other hand, enable the transfer of data to the physical assets, as well as operations invocation; said data and invocations can be understood by the physical assets on the basis of common semantic grounds and the functionality offered by the adapters, towards parameterization and control of the physical asset. Further, based on a change, devices can also be notified through the messaging infrastructure if an application wants to change something in the device.

Components view

The figure below shows the Ditto services (components), the externally provided and consumed API endpoints, the external dependencies and the relations of the services to each other.

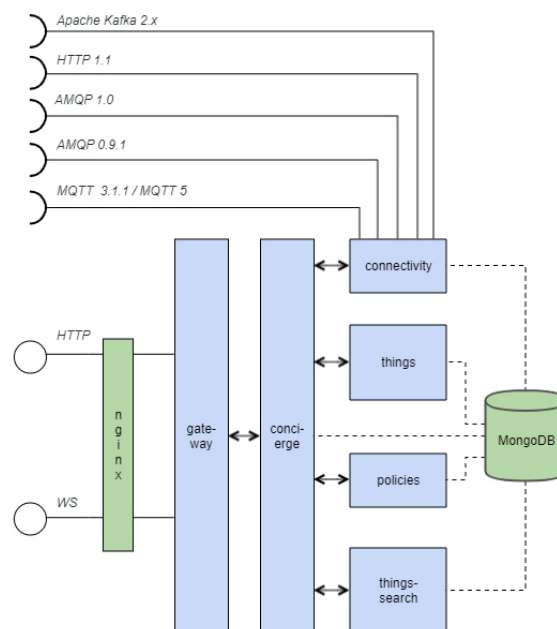


Figure 34 – Eclipse Ditto – Components view

The components have the following tasks:

- **Policies:** persistence of Policies;

- **Things:** persistence of Things and Features (next section);
- **Things-Search:** tracking changes to Things, Features, Policies and updating an optimized search index, also executing queries on this search index;
- **Concierge:** orchestrates and authorizes the backing persistence services;
- **Gateway:** provides the HTTP and WebSocket APIs, offered through a NGINX web server;
- **Connectivity:** sends Ditto Protocol messages to external message brokers and receives messages from them.

Persistence in Ditto is provided by a MongoDB database maintaining all digital twin state data, as well as platform configuration data (e.g., policies).

Modelling view

In the Ditto realm, digital twins are modelled as Things, which constitute very generic entities and are mostly used as “handles” for multiple features belonging to each such Thing.

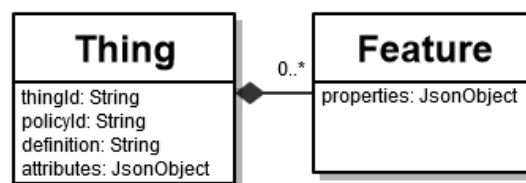


Figure 35 – Eclipse Ditto - Class diagram of Ditto's most basic entities in API version 2.

As shown in the figure above, a Thing is structured by the following elements:

- **Thing ID:** The unique identifier of a Thing.
- **Definition:** A Thing may contain a definition, used to link it to a corresponding model defining the capabilities/features of it. The definition can also be used to find Things.
- **Attributes:** Attributes describe the Thing in more detail and can be of any type. Attributes can also be used to find Things. Attributes are typically used to model rather static properties at the Thing level, in the sense that their values do not change as frequently as property values of Features.
- **Features:** A Thing may contain an arbitrary number of Features. A Feature is used to manage all data and functionality of a Thing that can be clustered in an outlined technical context (representing, for instance, the thermostat belonging to a physical device that is itself mirrored by a digital twin). For different contexts or aspects of a Thing different Features can be used which all belong to the same Thing and do not exist without this Thing.
- **Policy:** A Thing may contain a link to a Policy defining which authenticated subjects may READ and WRITE the Thing or even parts of it (hierarchically specified).

- **Metadata:** A Thing may contain additional metadata for all of its attributes and features, describing the semantics of the data or adding other useful information about the data points of the twin.

Regarding Features in particular, the data related to them are managed in the form of a list of properties. These properties can be categorized, e.g., to manage the status (e.g., on/off state), the configuration (e.g., the temperature set point of the thermostat) or any fault information. Each property itself can be either a simple/scalar value or a complex object. A feature may also include a list of, likewise managed, desired properties as a tool to represent the desired target state of the properties.

A feature may also define which behaviour/capabilities can be expected from it, in the form of events and operations. Events define data that are emitted by the device or entity (e.g., increase of water pH); this kind of data would need to be transmitted to interested AquaSPICE modules in a reliable way. An operation, on the other hand, represents a function that can be performed on a Digital Twin (e.g., turn on/off), hence trigger an action on a device. Events and operations are mapped to feature messages sent “to” or “from” a feature, according to the Ditto protocol; more specifically, a message sent to a feature has an operation as its subject, while a message sent from a feature has as subject an event.

In defining the attributes and features of AquaSPICE Things, the project will implement the specifications of ISO/DIS 23247-3 [6]. The table below summarises at a high level a minimum set of possible information types that the standard foresees for the corresponding digital twins.

Information element	Description	Mandatory (M) / Optional (O)
Identifier	Value used to uniquely identify an observable element	M
Characteristics	A typical or noticeable feature of an observable element. They mainly refer to static information that does not change during the process.	M
Schedule	Time information bound to a water-related process	M
Status	Situation of an observable element involved in a water-related process; typically, it may change during the process	M
Location	Geographical or relative location information of an observable element	M

Report	Description of activity done by or onto an observable element	M
Relationship	A connection information between two or more observable elements	M

Table 2 - Information attributes for the describing Things

The exact content and the specificities of the above information elements will depend on the types of things to be addressed on occasion and will be aligned with the descriptions provided for each entity. In this regard, and on the basis of the analysis of AquaSPICE case studies, the project also adopts the high-level categorisation of digital twins to be modelled proposed by ISO/DIS 23247, which is as follows:

- **Personnel:** Includes employees who are engaged directly or indirectly in the water purification processes.
- **Equipment:** Any physical element that carries out an operation directly or indirectly for a water purification process, e.g., a filter.
- **Material:** Physical matter that participates in the water purification process e.g., chemicals etc.
- **Process:** An observable physical operation within the water purification manufacturing, e.g., demineralization.
- **Facility:** This includes infrastructure that is related to or affects the process, e.g., a river, a lake.
- **Environment:** It includes necessary conditions that shall be supplied by facilities for the correct execution of a water purification process, e.g., temperature.
- **Product:** A desired output or by-product of the water purification process, or a group thereof.
- **Supporting document:** Any form of artifact that helps the applications of Digital Twin.

Interestingly, entities of the above types may form various relations while participating in the water purification process, while the final water quality is affected in the context of these relations. In this sense, the inclusions of such relations as part of the digital twin representation, as also foreseen by the standard, will be crucially useful to AquaSPICE, as they will be consistently taken into account and associated orchestrations of the necessary data exchanges among the Things involved.

Further, the same type of information may be modelled in Ditto either as an attribute or a feature depending on context; for instance, location is a static characteristic for a plant, however it constitutes a dynamic feature for manual water quality measurements. From another perspective, attributes/features can be defined in either absolute (e.g., geographical location of a plant) or relative (e.g., proximity to another entity) terms; the latter is supported by defining relationships among digital twins.

In both cases, attributes and features of the digital twins will be grounded to the ontology and data model specified under the activities of Task 1.4 “Data Models, Taxonomy and

Ontology for Water Use in the Process Industry” and Task 3.2 “Smart Network with Multi-Source Data Acquisition, Harmonisation and Processing Framework”. The ontology specified in deliverable D1.5, offers semantic grounding to the attributes and features of the digital twins, when viewed as information structures. For example, a temperature attribute will reference via an accompanying JSON-LD description the corresponding term in the Smart Data Models ontology (<https://smart-data-models.github.io/data-models/terms.jsonld#/definitions/temperature>). On the other hand, the data model specified in D3.2, is used as a domain model when defining the attributes and features of the digital twins, and acts as the foundation upon which interoperability between the digital twins and the rest of AquaSPICE modules is grounded.

Finally, AquaSPICE extends the set of information elements contained in Table 1 by query features, so as to address the requirement of accessing, through a Thing, data not stored within a Digital Twin; this refers mainly to historical data or other information that need to be retrieved on demand from AquaSPICE persistence, on-premise databases or systems or even external sources (e.g., energy cost data over previous days, weeks or months). This feature offers the necessary operations to be performed over the live channel on demand and make their results available as feature properties.

It is to be noted that further extensions are introduced, particularly in order to accommodate AquaSPICE platform-generated properties of digital twins; however, such extensions are out of the scope of this document and will be documented in future deliverables.