



ENERGY-AWARE FACTORY ANALYTICS FOR PROCESS INDUSTRIES

Deliverable D6.2

Data Collection Framework (Final Version)

Version
Version 1.5

Lead Partner
MAG

Date
29/04/2022

Project Name
FACTLOG – Energy-aware Factory Analytics for Process Industries

Call Identifier

H2020-NMBP-SPIRE-2019

Topic

DT-SPIRE-06-2019 - Digital technologies for improved performance in cognitive production plants

Project Reference

869951

Start dateNovember 1st, 2019**Type of Action**

IA – Innovation Action

Duration

42 Months

Dissemination Level

X	PU	Public
	CO	Confidential, restricted under conditions set out in the Grant Agreement
	CI	Classified, information as referred in the Commission Decision 2001/844/EC

Disclaimer

This document reflects the opinion of the authors only.

While the information contained herein is believed to be accurate, neither the FACTLOG consortium as a whole, nor any of its members, their officers, employees or agents make no warranty that this material is capable of use, or that use of the information is free from risk and accept no liability for loss or damage suffered by any person in respect of any inaccuracy or omission.

This document contains information, which is the copyright of FACTLOG consortium, and may not be copied, reproduced, stored in a retrieval system or transmitted, in any form or by any means, in whole or in part, without written permission. The commercial use of any information contained in this document may require a license from the proprietor of that information. The document must be referenced if used in a publication.

Executive Summary

The aim of WP6 is to implement the entire FACTLOG architecture and the system that integrates the enhanced cognitive twins and all tools and services over a cloud-based collaboration infrastructure, along the specifications reported in deliverable D1.3 “System Architecture and Technical Specifications” [3]. A key prerequisite in this direction is the development of a mediation middleware among the various components comprising the FACTLOG ecosystem, assigned with the interaction, coordination and orchestration of its components and operations. To address the above, FACTLOG proposes a data collection and integration 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.
- *Messaging and streaming*: It facilitates both asynchronous and point-to-point message exchange between system components, while also supporting streaming, thus 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 manufacturing 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 final version of the data collection and integration framework, reflecting the work performed in the context of the project Task 6.1 “Multi-modal Data Collection and Integration Framework”, and the outcome thereof. Closely following the analysis of the pilot cases and the associated requirements derived, the development of the rest of FACTLOG components, services and tools, as well as the application of the framework to the pilot cases, this deliverable describes the constituent modules and main functions of the proposed solution that render it the main integration enabler in FACTLOG platform.

Revision History

Revision	Date	Description	Organisation
1.0	20/01/2021	Overview of the data collection framework	MAG
1.1	26/01/2021	Incorporation of the digital twins chapter	MAG
1.2	29/01/2021	Interim version draft available to reviewers.	MAG
1.3	31/01/2021	Interim version submitted	MAG
1.4	26/04/2022	Final version available to reviewers	MAG
1.5	29/04/2022	Final version submitted	MAG

Contributors

Organisation	Author	E-Mail
MAG	Kostas Kalaboukas	kostas.kalaboukas@maggioli.it
MAG	Mariza Koukovini	mariza.koukovini@gmail.com
MAG	Aziz Mousas	az.mousas@gmail.com
MAG	Nikos Dellas	nikolaos.dellas@gmail.com
MAG	Eugenia Papagiannakopoulou	eugenia.papagiannakopoulou@gmail.com
MAG	Georgios Lioudakis	gelioud@ieee.org

Table of Contents

Executive Summary	3
Revision History	4
1 Introduction	7
1.1 Purpose and Scope	7
1.2 Relation with other Deliverables	7
1.3 Structure of the Document.....	7
1.4 Document updates	7
2 Data Collection Framework	9
3 Data Ingestion	11
3.1 Eclipse Hono	11
3.2 Apache NiFi	13
4 Data Management	17
4.1 Apache ActiveMQ.....	17
4.2 Apache Drill	18
5 Digital Twins	21
5.1 Eclipse Ditto.....	21
5.1.1 Functional view.....	21
5.1.2 Components view	25
5.2 Data model	26
References	31

List of Figures

Figure 1: The FACTLOG data collection framework	9
Figure 2: Eclipse Hono overview	11
Figure 3: Eclipse Hono architecture	12
Figure 4: Apache NiFi architecture	13
Figure 5: Data connector simple example.....	15
Figure 6: EvaluateJSONPath NiFi processor properties	15
Figure 7: ExecuteSQL NiFi processor properties.....	16
Figure 8: ActiveMQ - Publish/Subscribe.....	17
Figure 9: Apache ActiveMQ management console	18
Figure 10: Apache Drill interfaces	19
Figure 11: Apache Drill - Drillbit architecture.....	19
Figure 12: Apache Drill - Drillbit configuration.....	20
Figure 13: Apache Drill management console	20
Figure 14: Twin channel.....	22
Figure 15: Live channel.....	22
Figure 16: Eclipse Ditto REST API	25
Figure 17: Ditto components view.....	25
Figure 18: Class diagram of Ditto's most basic entities in API version 2.....	27
Figure 19: Cognitive functions of a digital twin.....	29
Figure 20: Implementation of a digital twin cognitive function in Apache NiFi.....	30

List of Tables

Table 1 - Information attributes for the describing MEs.....	27
--	----

1 Introduction

1.1 Purpose and Scope

The goal of WP6 “Integration and Toolset Creation” is to implement the FACTLOG architecture and setup the infrastructure for the integration of platform tools and services, and finally validate that the platform covers pilot requirements. This deliverable reports mainly on the results of Task 6.1 “Multi-model Data Collection and Integration Framework” presenting the data collection framework, that has been devised to support the FACTLOG 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.

1.2 Relation with other Deliverables

This deliverable evolves the main data communication and integration principles set forth within the deliverable D1.3 “System Architecture and Technical Specifications” [3]. In this direction, it provides, on the one hand, the technical specifications meant to implement the Message and Service Bus functionalities in terms of data ingestion and management, and, on the other, some insights on how the digital twins platform deployed for FACTLOG makes collected data accessible to all other interested applications and modules. Towards this goal, deliverable D1.2 “Cognitive Factory Framework” [2] has been used as a starting point in identifying required dependencies and interactions and deliverables of WP2 “Analytics and AI for Cognitive Factories”, WP3 “Enhanced Cognitive Twins”, WP4 “Knowledge Graph and Process Modelling”, WP5 “Robust Optimisation Methods” have been carefully analyzed for any additional requirements.

Certain aspects of the framework presented in this document are further elaborated in D6.5/D6.6 “Integrated Package and Platform (Interim / Final Version), which signify the 1st and 2nd pilot iterations, respectively, and, as such, the application of the proposed data collection mechanisms.

1.3 Structure of the Document

The introduction first outlines what the purpose of this document is and the areas it covers in Section 1.1, and then lists all the deliverables related to this document, serving either as its input or representing future work based upon its content, in Section 1.2. An overview of the data collection framework is presented in Section 2. The technologies used in the Data Ingestion layer of the framework are presented in Section 3. Namely, Eclipse Hono for the ingestion of IoT device data (Section 3.1), and Apache Nifi for designing and executing ingestion flows from legacy data sources (Section 3.2). The technologies used in the Data Management layer of the framework are presented in Section 4. Namely, Apache ActiveMQ is used as the message broker between platform components realizing the publish/subscribe communication paradigm. For querying the attached to the platform data sources, Apache Drill is leveraged providing an SQL query interface. Finally, Section 5 presents the Digital Twins layer, where Eclipse Ditto is employed providing APIs for managing and interacting with digital twins.

1.4 Document updates

The document contains the following updates in relation to the interim version:

- Section 4.1: includes a figure of the management console of the message broker, which shows the queues and the topics that are used by the system.
- Section 4.2: includes a figure of the management console of the distributed query engine, which is used for monitoring and optimization purposes.
- Section 5.1: includes a figure with the REST API of Eclipse Ditto
- Section 5.2: presents the implementation of cognitive functions in Apache NiFi as part of the digital twins

2 Data Collection Framework

Effective communication of data coming from the plant floor and routing to the appropriate components is vital for the operation of the FACTLOG cognition process. To this end, the data collection framework facilitates the realization of the digital twins paradigm offering a toolset for connecting IIoT devices and data sources to their digital representation, as well as APIs to applications which leverage digital twins to enhance their operation.

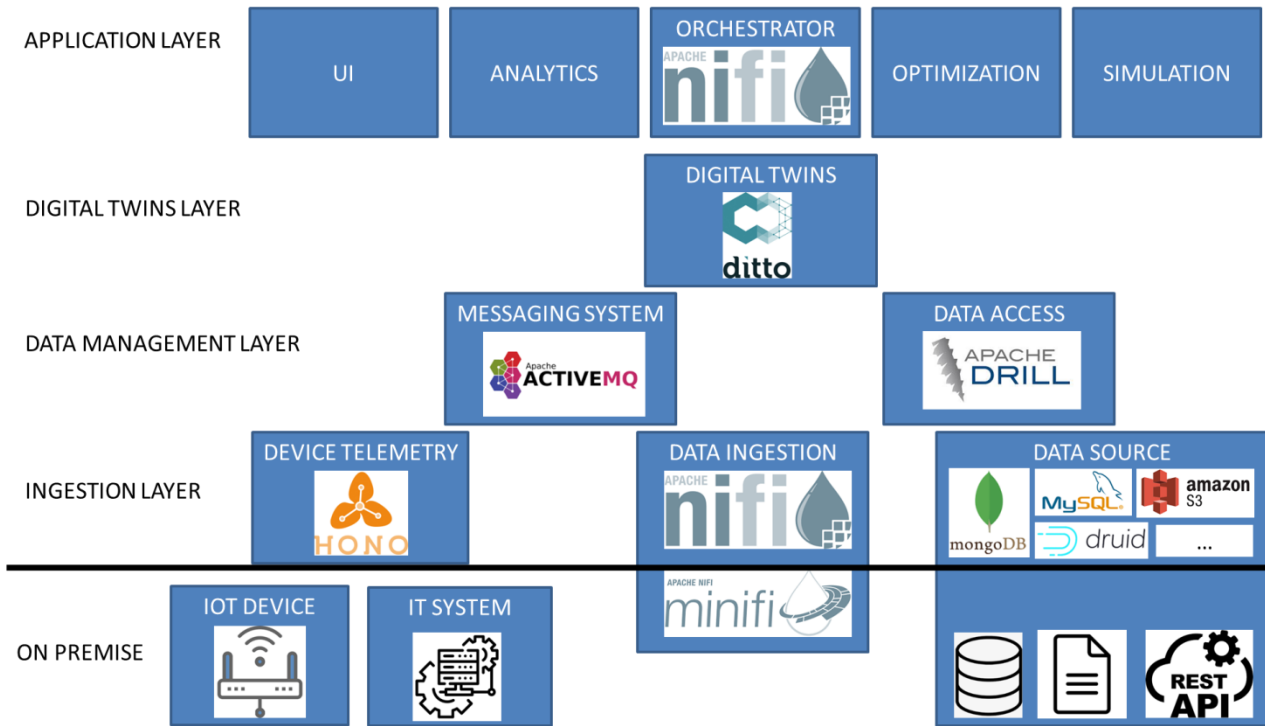


Figure 1: The FACTLOG data collection framework

The figure above presents the layered architecture that the FACTLOG data collection framework follows:

- The **factory layer** holds the entities that constitute the primary data producers in the FACTLOG platform. IIoT devices and sensors generate readings that are communicated to the platform either directly to the FACTLOG 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 is 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 FACTLOG platform adopts the Apache MiNiFi solution, which acts as a lightweight on premise agent of the data ingestion layer.

- The **data management layer**, following the ingestion of data from the factory layer, is responsible for communicating the data to the appropriate components. The FACTLOG platform adopts the publish/subscribe paradigm and, using Apache ActiveMQ as the message broker, enables platform components to consume data coming from the shop floor, as well as exchange data in an asynchronous manner. Moreover, the data management layer provides an interface for accessing the data sources that are 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's 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 cognition process, 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 data communication layer. Lastly, among the applications, the orchestrator has a key role in the FACTLOG platform, since it drives the cognition process by connecting analytics, optimization and simulation services to the underlying digital twins. Here, the Apache NiFi dataflow framework is used, allowing flexible integration between the components and continuous monitoring of its status.

The following chapters present in more detail the components that are involved in each layer along with their capabilities. Starting with the data ingestion layer, where data is captured, continuing with the data management layer, where data is communicated to platform components, and finishing with the digital twins layer, which holds the state of the system enhanced with cognitive functionalities.

3 Data Ingestion

The main goal of the data ingestion layer is the efficient capturing and delivery of plant floor data (machines, materials, resource consumption) and manufacturing chain data (capacity, inventory, lead time) to the data management layer. To this end, the FACTLOG platform distinguishes between IIoT devices and legacy data sources and offers tailor-made solutions for handling data ingestion from them. For the former, it leverages the Eclipse Hono¹ solution for capturing device telemetry data. While for the latter, it employs the Apache NiFi² dataflow management solution to design, execute and monitor the data ingestion processes. The sections below present the basic concepts and features of both solutions along with the way that they are used in FACTLOG.

3.1 Eclipse Hono

Given the communication protocol landscape in the domain of IIoT, 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 offers a uniform interface for IoT applications to access, command and control IIoT devices, independently from the communication protocol that they use.

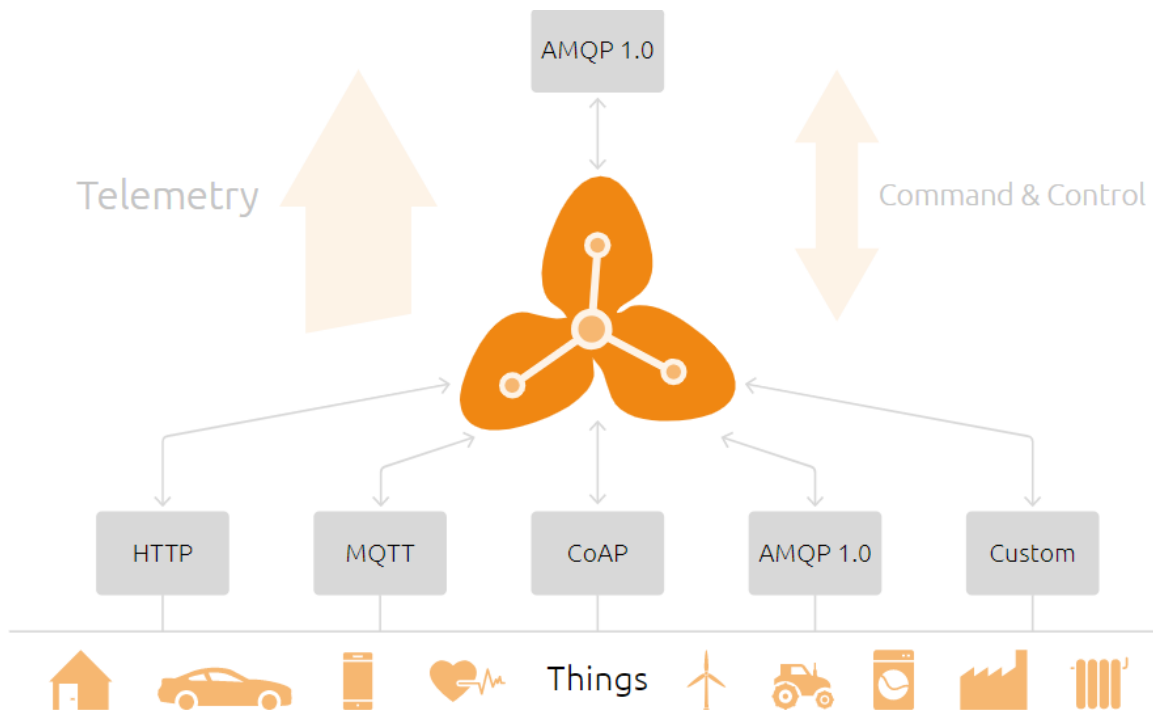


Figure 2: Eclipse Hono overview

¹ <https://www.eclipse.org/hono/>

² <https://nifi.apache.org/>

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 of 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 Grafana³ for visualizing and Prometheus⁴ for collecting the device message logs.

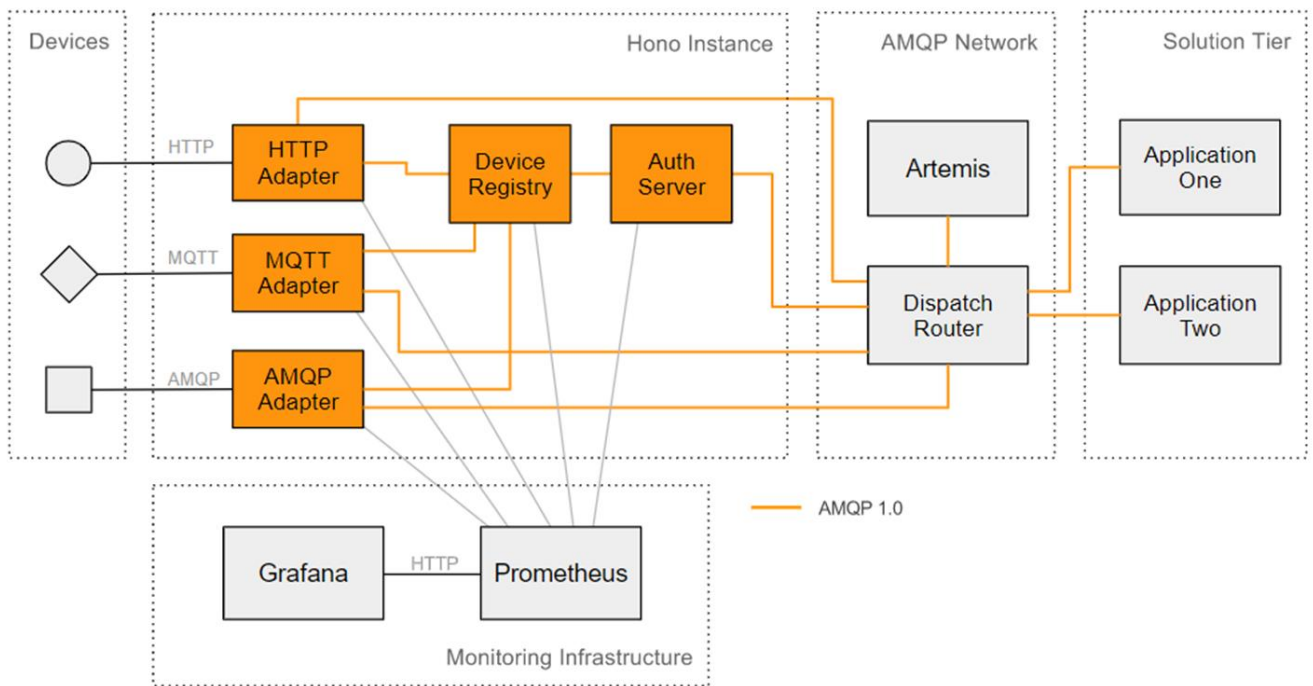


Figure 3: Eclipse Hono architecture

In FACTLOG each device that needs to be directly connected to the platform can be registered to the Device Registry, and 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 can be used to push data to the platform.

³ <https://grafana.com/>

⁴ <https://prometheus.io/>

3.2 Apache NiFi

The ingestion of data from legacy data sources, e.g., databases, files and web services, poses to the FACTLOG 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 machine maintenance hours might be required in order to compute optimal production schedules.

In order to achieve scenarios like the one described above, the FACTLOG 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.

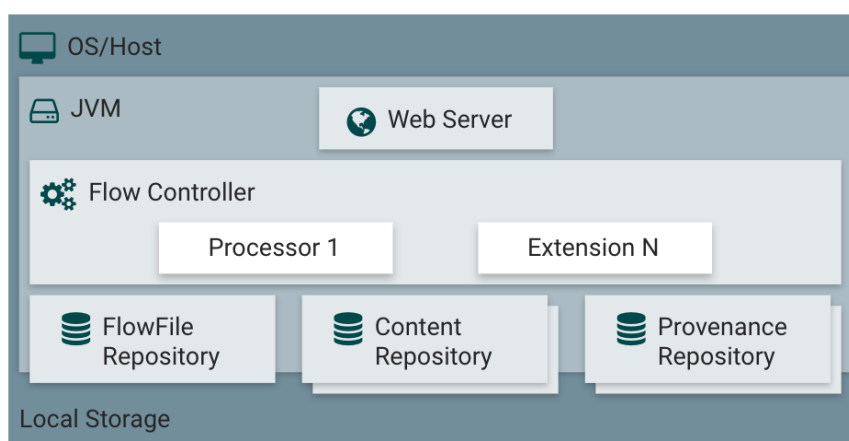


Figure 4: Apache NiFi architecture⁵

⁵ <https://nifi.apache.org/docs/nifi-docs/html/overview.html>

As shown in Figure 4 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**. Apache 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 NiFi's 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 an order entry based on the order identifier (orderId) is shown in Figure 5. The 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 order identifier (for example, “{‘orderId’:1}”) and produces a JSON object containing all database table entry columns that are stored in a local file.

The GenerateFlowFile processor is used for storing the JSON String to a FlowFile. Then, the EvaluateJSONPath processor is used for extracting the “orderId” from the JSON String and store it to the “orderId” attribute (cf. Figure 6). The ExecuteSQL processor performs the actual communication to the database utilizing the “orderId” attribute (cf. Figure 7). The SplitAvro processor 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 is stored into a local file by the PutFile processor.

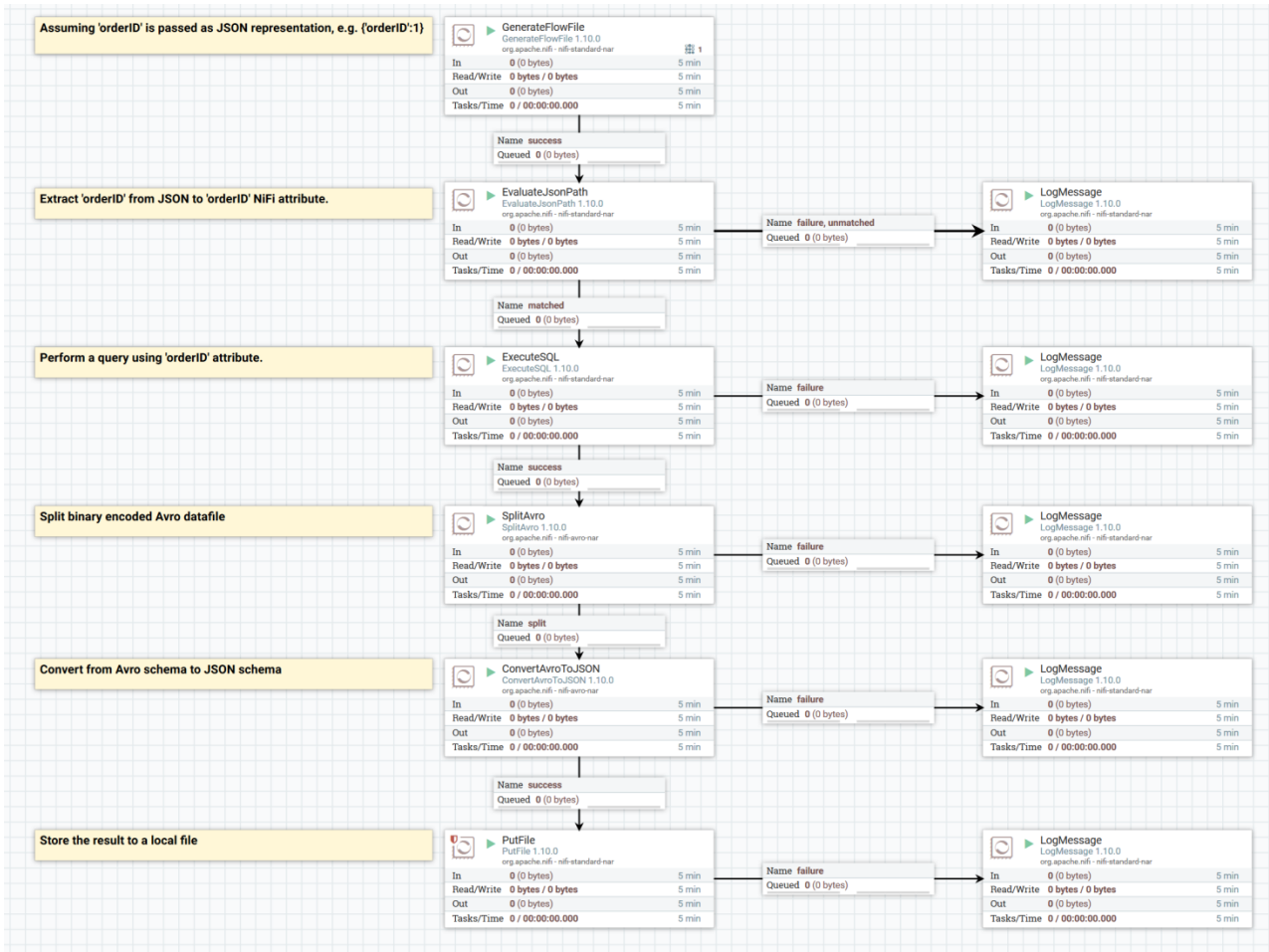


Figure 5: Data connector simple example

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
orderId	? \$.orderId 🗑️

Figure 6: EvaluateJSONPath NiFi processor properties

Configure Processor

Stopped

SETTINGS | SCHEDULING | **PROPERTIES** | COMMENTS

Required field +

Property	Value
Database Connection Pooling Service	DBCPConnectionPool →
SQL Pre-Query	No value set
SQL select query	
SQL Post-Query	
Max Wait Time	
Normalize Table/Column Names	
Use Avro Logical Types	
Compression Format	
Default Decimal Precision	
Default Decimal Scale	
Max Rows Per Flow File	
Output Batch Size	

EL ✓ PARAM ✓

```

1 select
2   '${payweight}' as payweight,
3   '${order_type}' as order_type,
4   '${orderTypeLocationID}' as orderTypeLocationID,
5   '${issue_time}' as issue_time,
6   '${ready_time}' as ready_time,
7   '${desiredDeliveryTimeFrame}' as desiredDeliveryTimeFrame,
8   '${importance}' as importance
9 from Order
10 where orderID = ${orderID}
    
```

Set empty string

CANCEL OK LY

Figure 7: ExecuteSQL NiFi processor properties

4 Data Management

The main goal of the data management layer is the efficient routing of plant floor and manufacturing chain data, as well as FACTLOG component generated data to the digital twins and application layer. To this end, the FACTLOG platform distinguishes between messaging and on demand data access and offers tailor-made solutions for both cases. For the former, it leverages the Apache ActiveMQ⁶ solution for routing data to FACTLOG components. While for the latter, it employs the Apache Drill⁷ distributed query engine to provide a unified SQL query interface to connect 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 are used in FACTLOG.

4.1 Apache ActiveMQ

The role of the Apache ActiveMQ message broker is critical to the platform, since FACTLOG, 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 plant floor data (machines, materials, resource consumption), manufacturing chain data (capacity, inventory, lead time), to data generated by FACTLOG components (digital twin state change, analytics, 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 FACTLOG use cases, data will be routed, as shown in Figure 8, 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.

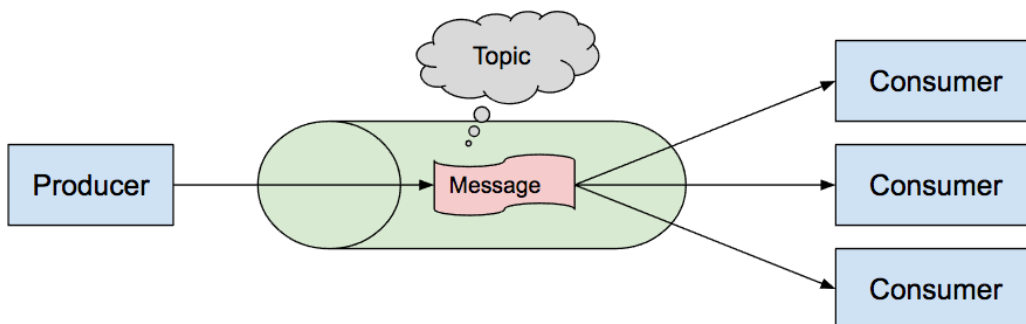


Figure 8: ActiveMQ - Publish/Subscribe

⁶ <https://activemq.apache.org/>

⁷ <https://drill.apache.org/>

The following figure presents the management console of the Apache ActiveMQ message broker, which lists all the available queues and topics that are used by the system components. It must be highlighted here that outside of the DITTO-IN and DITTO-OUT message queues, which are utilized for the communication of the digital twins layer with the data management layer, the rest of the addresses follow typically the “event/TENANT/COMPONENT/queue” structure.

manage	ID	Name	Address	Routing Type	Filter	Durable	Max Consumers	Purge On No Co...	Consumer Count	Rate	Message Count
attributes operati...	1203176	\$sys.mqgt.retain...	\$sys.mqgt.retain...	MULTICAST		true	-1	false	0	0.0	1
attributes operati...	1456936	\$sys.mqgt.retain...	\$sys.mqgt.retain...	MULTICAST		true	-1	false	0	0.0	1
attributes operati...	1397528	\$sys.mqgt.retain...	\$sys.mqgt.retain...	MULTICAST		true	-1	false	0	0.0	1
attributes operati...	272731271274	DITTO-IN	DITTO-IN	ANYCAST		true	-1	false	1	0.0	0
attributes operati...	272731271271	DITTO-OUT	DITTO-OUT	ANYCAST		true	-1	false	1	0.0	0
attributes operati...	272731294355	BRC-activemq-brc...	event/BRC/NDAT...	MULTICAST	__AMQ_CID<=>BR...	true	1	false	1	0.0	0
attributes operati...	272731295007	BRC-activemq-brc...	event/BRC/NDAT...	MULTICAST	__AMQ_CID<=>BR...	true	1	false	1	0.0	0
attributes operati...	803027	event/BRC/analy...	event/BRC/analy...	ANYCAST		true	-1	false	0	0.0	1
attributes operati...	272731271126	event/BRC/db/*c...	event/BRC/db/*c...	ANYCAST		true	-1	false	1	0.0	0
attributes operati...	272731271092	event/BRC/ditto	event/BRC/ditto	ANYCAST		true	-1	false	1	0.0	0
attributes operati...	272731271090	event/BRC/optimi...	event/BRC/optimi...	ANYCAST		true	-1	false	1	0.0	0
attributes operati...	272731271205	event/BRC/query...	event/BRC/query...	ANYCAST		true	-1	false	1	0.0	0
attributes operati...	272731271328	event/BRC/simula...	event/BRC/simula...	ANYCAST		true	-1	false	1	0.0	0
attributes operati...	272744959988	CONT-analytics-in...	event/CONT/anal...	MULTICAST	__AMQ_CID<=>CO...	true	1	false	0	0.0	0
attributes operati...	272810969884	event/CONT/anal...	event/CONT/anal...	ANYCAST		true	-1	false	1	0.0	0
attributes operati...	272810937205	event/CONT/anal...	event/CONT/anal...	ANYCAST		true	-1	false	1	0.0	0
attributes operati...	268436287518	CONT-db-ingestio...	event/CONT/db/**	MULTICAST	__AMQ_CID<=>CO...	true	1	false	1	0.0	0
attributes operati...	272809425775	event/CONT/db/L...	event/CONT/db/L...	ANYCAST		true	-1	false	0	0.0	7
attributes operati...	272809426133	event/CONT/db/L...	event/CONT/db/L...	ANYCAST		true	-1	false	0	0.0	7
attributes operati...	272809447845	event/CONT/db/P...	event/CONT/db/P...	ANYCAST		true	-1	false	0	0.0	1204
attributes operati...	272809447695	event/CONT/db/P...	event/CONT/db/P...	ANYCAST		true	-1	false	0	0.0	3
attributes operati...	373900447331	event/CONT/db/P...	event/CONT/db/P...	ANYCAST		true	-1	false	0	0.0	67

Figure 9: Apache ActiveMQ management console

4.2 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 stored in MongoDB with machine maintenance schedules stored as files in a directory.

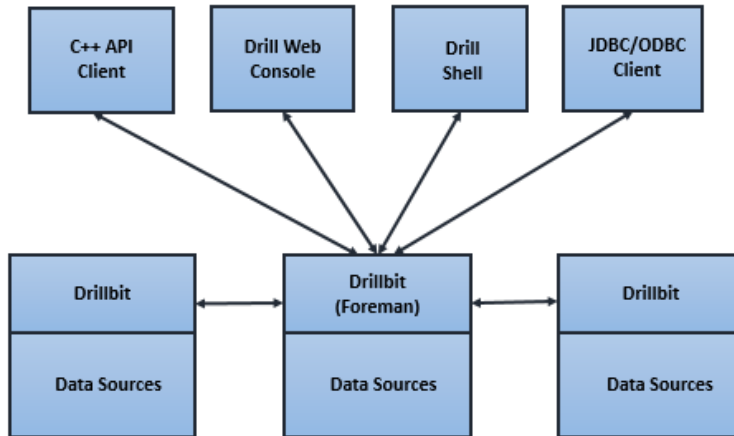


Figure 10: Apache Drill interfaces

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.

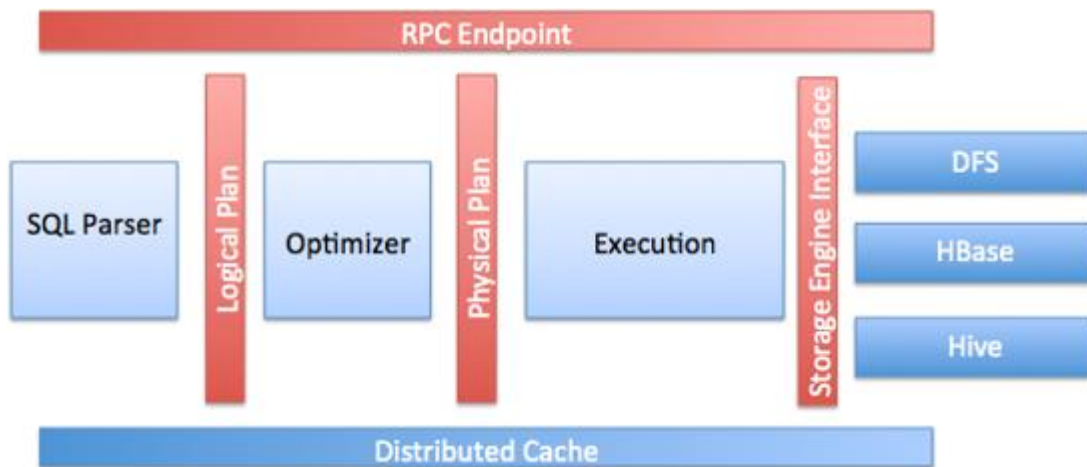


Figure 11: Apache Drill - Drillbit architecture

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 12: Apache Drill - Drillbit configuration

The management console of Apache Drill as shown in the following figure offers additional information regarding the execution of the queries. The total cost of a query is calculated and the system administrator may use this information to optimize performance of the queries.

The screenshot shows the Apache Drill management console interface. At the top, there are navigation tabs for 'Query', 'Profiles', 'Storage', 'Metrics', 'Threads', and 'Logs'. Below this, the 'Query and Planning' section is active, displaying a SQL query: `select timestamp, value from mongo.tuprak.unit_impurities where thingId='TUPRA3/Crude_Distillation_Unit-2' and tag='O2CP051' and timestamp > 1420063200 and timestamp < 1448651690,648`. Below the query, the 'Query Profile: 1dbb9254-9a93-16e2-d898-95d4f4a100f9' is shown. The 'Operator Profiles' section is expanded to show an 'Overview' table with the following data:

Operator ID	Type	Avg Setup Time	Max Setup Time	Avg Process Time	Max Process Time	Min Wait Time	Avg Wait Time	Max Wait Time	% Fragment Time	% Query Time	Rows	Avg Peak Memory	Max Peak Memory
00-xx-00	SCREEN	0.000s	0.000s	0.000s	0.000s	0.000s	0.000s	0.000s	0.81%	0.81%	354	-	-
00-xx-01	PROJECT	0.000s	0.000s	0.000s	0.000s	0.000s	0.000s	0.000s	1.01%	1.01%	354	72KB	72KB
00-xx-02	MONGO_SUB_SCAN	0.000s	0.000s	0.003s	0.003s	0.000s	0.000s	0.000s	98.18%	98.18%	354	72KB	72KB

Figure 13: Apache Drill management console

5 Digital Twins

In FACTLOG, all access to the production infrastructure and the data associated with it is based on the digital twin paradigm. In this regard, the technology of choice for accommodating the specifications presented in D1.3 [3] has been to use Eclipse Ditto, an open-source framework for creating and managing digital twins in the IoT. Ditto in this context acts as a digital twin middleware, capable of:

- providing a digital representation of real physical devices, offering a consistent view across a variety thereof;
- acting as broker for communicating with assets;
- facilitating representation of related processes or services.

This section provides a brief overview of Ditto core features with respect to the project needs, as well as of the associated data model that facilitate the exploitation of the collected data according to the FACTLOG digital twin approach.

5.1 Eclipse Ditto

At the core of Ditto lies a data model (to be elaborated in section 5.2) 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.

5.1.1 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.

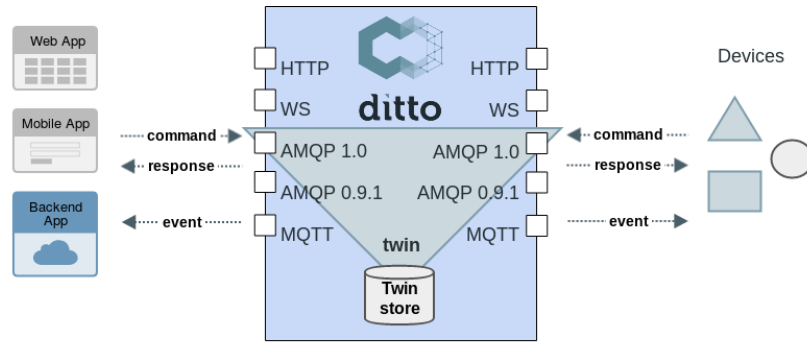


Figure 14: Twin channel

However, historical values are not persisted in Ditto. In order to cover the need to access historical data, FACTLOG data collection framework 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.

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.

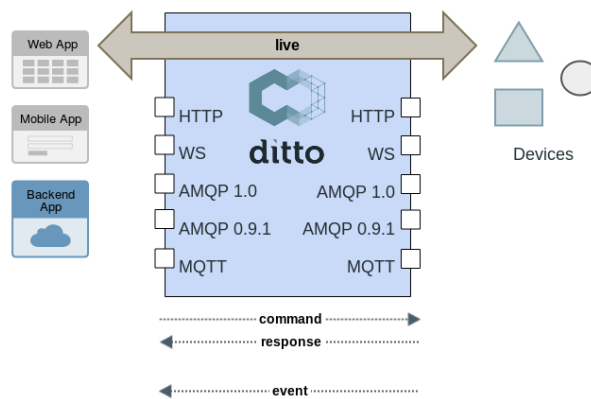


Figure 15: Live channel

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 RQL⁸ 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 FACTLOG can be used for a variety of purposes, according to requirements posed by the cognitive modules as well as the front end, in order, for example, to create a dashboard to show the real-time data of machines.

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 is more fitting to 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 can 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

⁸ <https://github.com/persvr/rql>

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 FACTLOG this feature is used in order to integrate the Ditto instance with data coming from the data ingestion and management layers.

On the basis of the above APIs, the interactions that are supported by Ditto in FACTLOG 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 wiring, 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 FACTLOG cognition domain as well as the business domain, i.e., any interested applications (e.g., predictive maintenance), 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.

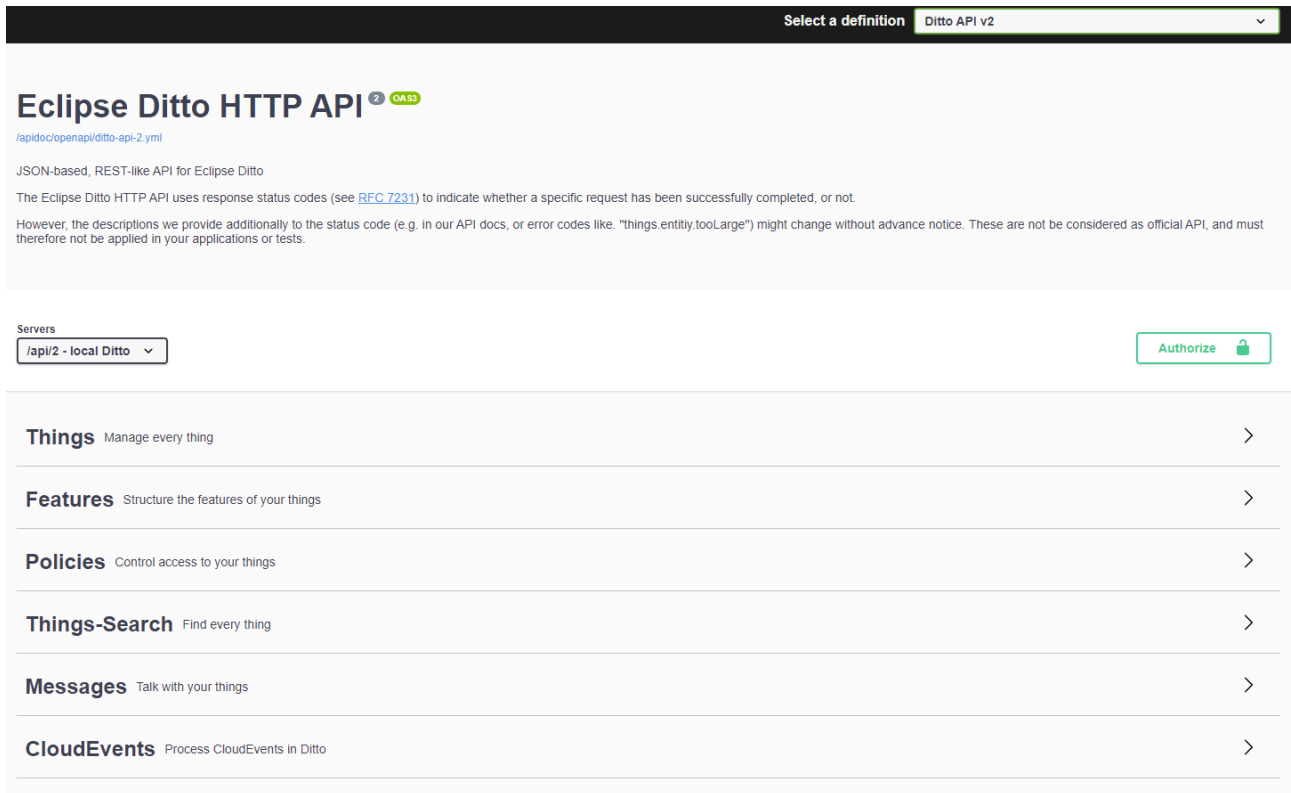


Figure 16: Eclipse Ditto REST API

5.1.2 Components view

Figure 17 shows the Ditto services (components), the externally provided and consumed API endpoints, the external dependencies and the relations of the services to each other.

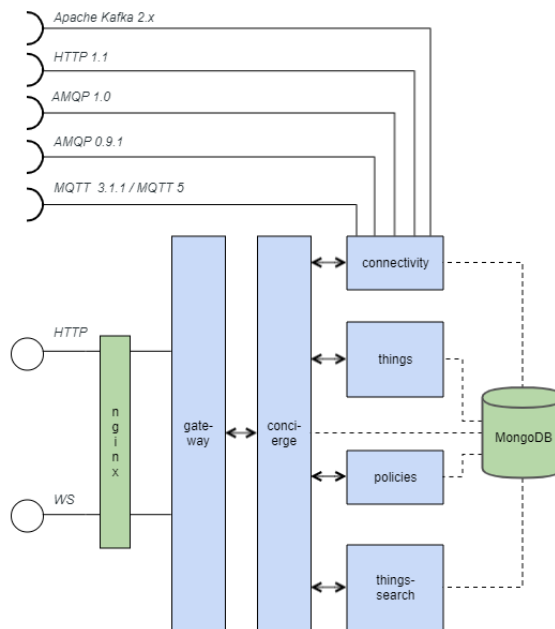


Figure 17: Ditto components view

The components have the following tasks:

- *Policies*: persistence of Policies;
- *Things*: persistence of Things and Features (cf. Section 5.2);
- *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 NGINX⁹ 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).

5.2 Data model

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. As shown in Figure 18, 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. In FACTLOG, Things definitions will follow the semantic descriptions provided for each corresponding entity by the Knowledge Graph.
- *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 amount 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

⁹ <https://www.nginx.com/>

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., machine breakdown); this kind of data would need to be transmitted to interested FACTLOG 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.

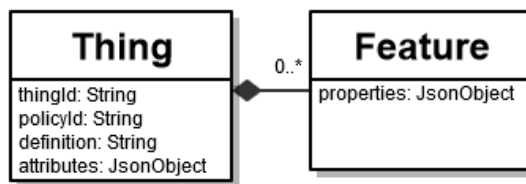


Figure 18: Class diagram of Ditto's most basic entities in API version 2.

In defining the attributes and features of FACTLOG Things, the project will implement the specifications of ISO/DIS 23247-3 [1], which address the particularities of the manufacturing domain. Table 1 summarises at a high level a minimum set of possible information types that the standard foresees for the corresponding digital twins.

Table 1 - Information attributes for the describing MEs

Information element	Description	Mandatory (M) Optional (O)
Identifier	Value used to uniquely identify an observable manufacturing element	M
Characteristics	A typical or noticeable feature of an observable manufacturing element. They mainly refer to static information that does not change during manufacturing ¹⁰ .	M
Schedule	Time information bound to a manufacturing process	M
Status	Situation of an observable manufacturing element involved in a manufacturing process; typically it may change during the process ¹¹ .	M

¹⁰ For Ditto, they may be attributes or desired properties.

¹¹ To be mapped to Ditto Features.

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

The exact content and the specificities of the above information elements will depend on the types of MEs to be addressed on occasion and will be aligned with the descriptions provided for each entity by the KG. In this regard, and on the basis of the analysis of FACTLOG use cases, 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 manufacturing processes.
- *Equipment*: Any physical element that carries out an operation directly or indirectly for a manufacturing process, e.g., a machine, tank or trailer.
- *Material*: Physical matter that becomes a part or the whole of a product e.g., bars, coils, feedstock, scrap, etc., or is used to aid manufacturing processes, e.g., cleaning fluid, coolant, etc.
- *Process*: An observable physical operation within manufacturing, e.g., the LPG distillation process.
- *Facility*: This includes infrastructure that is related to or affects manufacturing, e.g., a warehouse, a product laydown area.
- *Environment*: It includes necessary conditions that shall be supplied by facilities for the correct execution of a manufacturing process, e.g., air temperature.
- *Product*: A desired output or by-product of manufacturing process, or a group thereof (e.g., an order).
- *Supporting document*: Any form of artifact that helps the applications of Digital Twin for manufacturing.

Interestingly, entities of the above types may form various relations while participating in manufacturing processes (e.g., electronic equipment constituting a machine, machines participating in the same process, processes that if combined lead to a specific product), while the final production outcomes are 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, are crucially useful to FACTLOG and are consistently taken into account in cognitive functions 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 or a room (with reference to the building layout), however it constitutes a dynamic feature for a crane moving products and materials between machines. From another perspective, attributes/features can be defined in either absolute (e.g., geographical location of a plant or vehicle) or relative (e.g., proximity to another entity) terms; the latter is supported by

defining relationships among digital twins. Figure 19 presents the cognitive functions that enhance the digital twins as they are implemented in Apache NiFi.

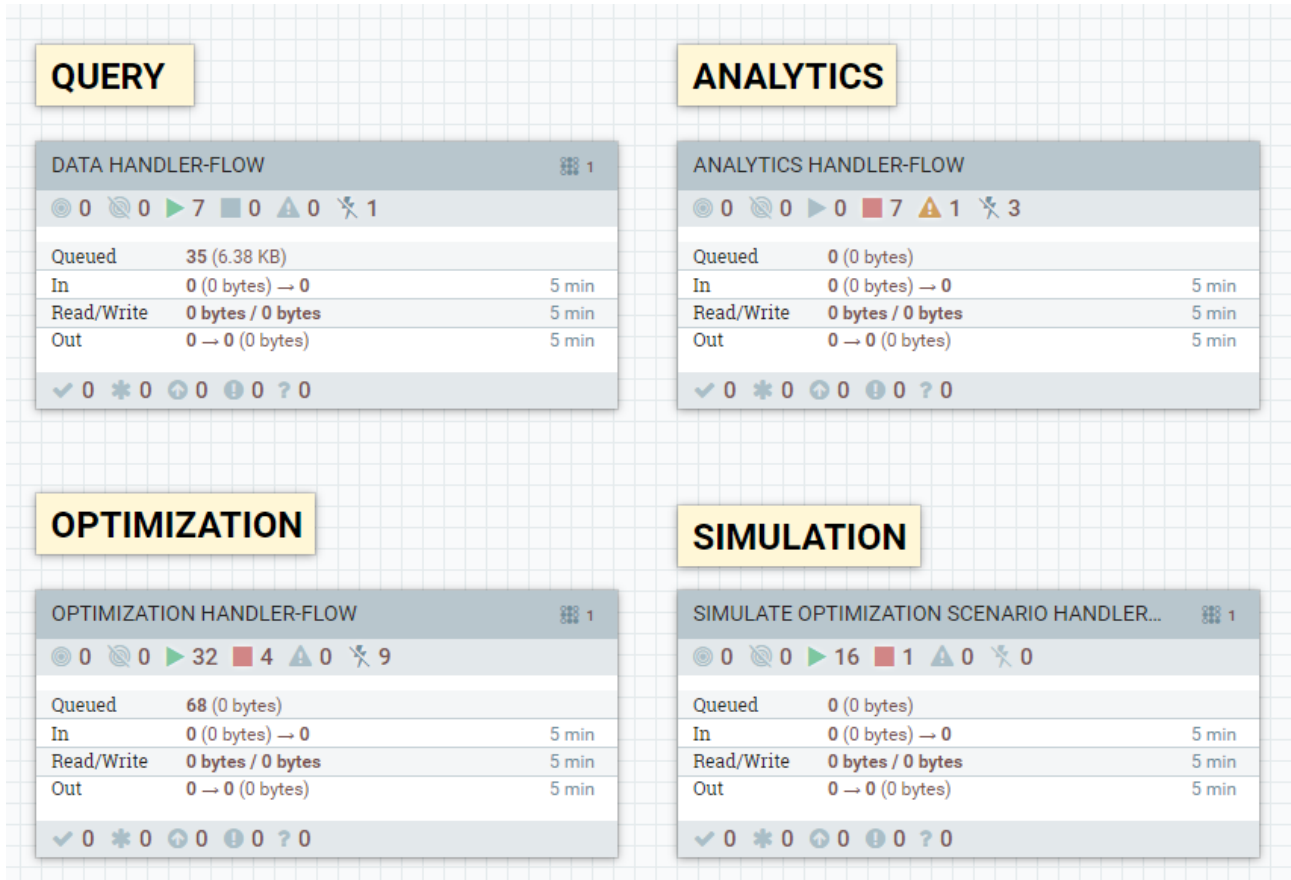


Figure 19: Cognitive functions of a digital twin

Finally, of particular interest is the operation that handles requests for 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 FACTLOG persistence, on-premise databases or systems or even external sources (e.g., weather or financial data over previous days, weeks or months). This feature operation is accessed over the live channel (cf. Section 5.1.1) on demand and the corresponding implementation is shown in Figure 20.

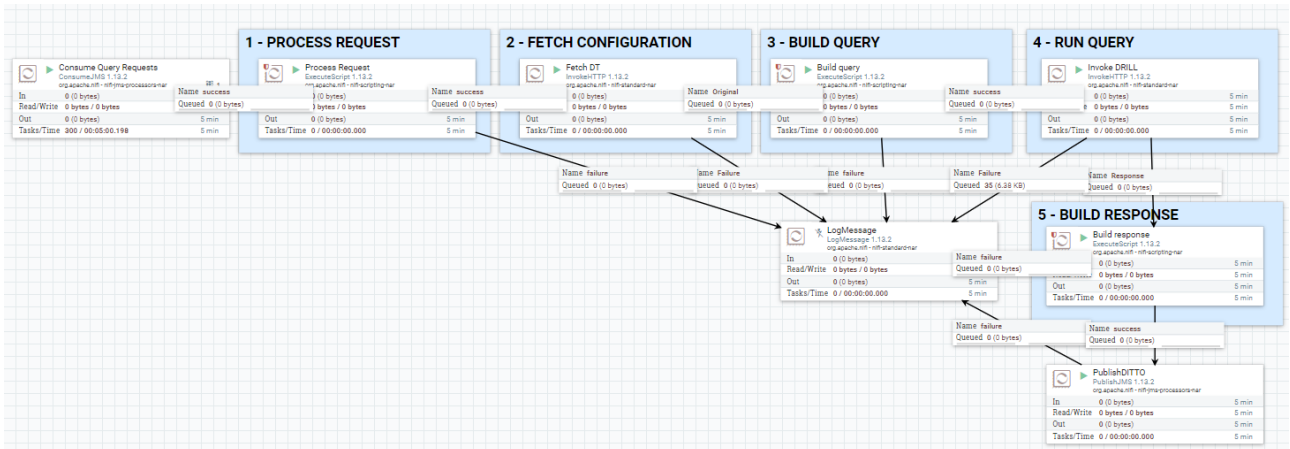


Figure 20: Implementation of a digital twin cognitive function in Apache NiFi

References

- [1] FACTLOG Deliverable D1.1 (2020). Reference Scenarios, KPIs and Datasets.
- [2] FACTLOG Deliverable D1.2 (2020). Cognitive Factory Framework.
- [3] FACTLOG Deliverable D1.3 (2020). System Architecture and Technical Specifications.
- [4] FACTLOG Deliverable D2.1 (2020). Analytics System Requirements and Design Specification
- [5] ISO/DIS 23247-3, “Automation systems and integration — Digital Twin framework for manufacturing — Part 3: Digital representation of manufacturing elements”, Draft International Standard, Reference number: ISO/DIS 23247-3:2020(E).