



European
Commission

Horizon 2020
European Union funding
for Research & Innovation

Big Data technologies and extreme-scale analytics



Multimodal Extreme Scale Data Analytics for Smart Cities Environments

D2.2: Management and distribution Toolkit – initial version[†]

Abstract: This document describes the initial version of the MARVEL management and distribution toolkit, which includes (i) the data management and distribution toolkit, and (ii) the Data Corpus-as-a-service for smart cities. The core involved components in the first part are the StreamHandler, DatAna, HDD, and DFB components and in the second part the Data Corpus component. Each component is presented in-depth in terms of novelty, design methodology, implementation, relation to other components, demonstration, and performance. Additionally, the current components' status within the E2F2C continuum is reviewed, along with the future directions of each component.

Contractual Date of Delivery	30/06/2022
Actual Date of Delivery	30/06/2022
Deliverable Security Class	Public
Editor	<i>Theofanis Raptis (CNR)</i>
Contributors	CNR, ITML, INTRA, ATOS, STS, UNS
Quality Assurance	<i>Grigoris Kalogiannis (STS)</i> <i>Tassos Kanellos (ITML)</i>

[†] The research leading to these results has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957337.

The *MARVEL* Consortium

Part. No.	Participant organisation name	Participant Short Name	Role	Country
1	FOUNDATION FOR RESEARCH AND TECHNOLOGY HELLAS	FORTH	Coordinator	EL
2	INFINEON TECHNOLOGIES AG	IFAG	Principal Contractor	DE
3	AARHUS UNIVERSITET	AU	Principal Contractor	DK
4	ATOS SPAIN SA	ATOS	Principal Contractor	ES
5	CONSIGLIO NAZIONALE DELLE RICERCHE	CNR	Principal Contractor	IT
6	INTRASOFT INTERNATIONAL S.A.	INTRA	Principal Contractor	LU
7	FONDAZIONE BRUNO KESSLER	FBK	Principal Contractor	IT
8	AUDEERING GMBH	AUD	Principal Contractor	DE
9	TAMPERE UNIVERSITY	TAU	Principal Contractor	FI
10	PRIVANOVA SAS	PN	Principal Contractor	FR
11	SPHYNX TECHNOLOGY SOLUTIONS AG	STS	Principal Contractor	CH
12	COMUNE DI TRENTO	MT	Principal Contractor	IT
13	UNIVERZITET U NOVOM SADU FAKULTET TEHNICKIH NAUKA	UNS	Principal Contractor	RS
14	INFORMATION TECHNOLOGY FOR MARKET LEADERSHIP	ITML	Principal Contractor	EL
15	GREENROADS LIMITED	GRN	Principal Contractor	MT
16	ZELUS IKE	ZELUS	Principal Contractor	EL
17	INSTYTUT CHEMII BIOORGANICZNEJ POLSKIEJ AKADEMII NAUK	PSNC	Principal Contractor	PL

Document Revisions & Quality Assurance

Internal Reviewers

1. Grigoris Kalogiannis (STS)
2. Tassos Kanellos (ITML)

Revisions

Version	Date	By	Overview
1.0.0	29/06/2022	CNR	Final draft
0.9.3	28/06/2022	ITML, STS, CNR	Corrected proof
0.9.2	25/06/2022	FORTH, CNR	Corrected proof
0.9.1	17/06/2022	CNR	Pre-final draft
0.9.0	16/06/2022	ITML, STS	Post-reviewed draft approval
0.8.0	13/06/2022	CNR	Post-reviewed draft
0.7.1	10/06/2022	STS	Reviewed draft
0.7.0	09/06/2022	ITML	Reviewed draft
0.6.0	27/05/2022	CNR	Second draft for reviewing
0.5.3	23/05/2022	STS	Section 3 inputs
0.5.2	20/05/2022	CNR, INTRA, ITML, ATOS	Sections 2, 4, 5 refinements
0.5.1	17/05/2022	CNR	First draft
0.5.0	13/05/2022	CNR, INTRA, ITML, ATOS	Sections 2, 4, 5 inputs
0.4.0	21/04/2022	CNR	Final draft ToC
0.3.1	21/04/2022	UNS	Comments on the ToC
0.3.0	21/04/2022	CNR	Third draft ToC
0.2.1	19/04/2022	UNS, STS	Comments on the ToC
0.2.0	18/04/2022	CNR	Second draft ToC
0.1.1	14/04/2022	UNS	Comments on the ToC
0.1.0	31/03/2022	CNR	First draft ToC

Disclaimer

The work described in this document has been conducted within the MARVEL project. This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957337. This document does not reflect the opinion of the European Union, and the European Union is not responsible for any use that might be made of the information contained therein.

This document contains information that is proprietary to the MARVEL Consortium partners. Neither this document nor the information contained herein shall be used, duplicated or communicated by any means to any third party, in whole or in parts, except with prior written consent of the MARVEL Consortium.

Table of Contents

LIST OF TABLES.....	6
LIST OF FIGURES.....	7
LIST OF ABBREVIATIONS.....	8
EXECUTIVE SUMMARY	10
1 INTRODUCTION.....	11
1.1 SCOPE AND PURPOSE	11
1.2 CONTRIBUTION TO WP2 AND PROJECT OBJECTIVES.....	11
1.3 RELATION TO OTHER WPS, DELIVERABLES, AND ACTIVITIES.....	12
2 COMPONENTS OF THE DMP TOOLKIT.....	14
2.1 DATANA	15
2.1.1 <i>Progress beyond the state-of-the-art</i>	15
2.1.2 <i>Design methodology</i>	15
2.1.3 <i>Implementation</i>	18
2.1.4 <i>Connection to other components</i>	21
2.1.5 <i>Demonstration</i>	21
2.1.6 <i>Performance</i>	22
2.2 DFB	24
2.2.1 <i>Progress beyond the state-of-the-art</i>	24
2.2.2 <i>Design methodology</i>	26
2.2.3 <i>Implementation</i>	27
2.2.4 <i>Connection to other components</i>	29
2.2.5 <i>Demonstration</i>	29
2.2.6 <i>Performance</i>	29
2.3 STREAMHANDLER.....	31
2.3.1 <i>Progress beyond the state-of-the-art</i>	34
2.3.2 <i>Design methodology</i>	35
2.3.3 <i>Implementation</i>	36
2.3.4 <i>Connection to other components</i>	37
2.3.5 <i>Performance</i>	37
2.4 HDD	38
2.4.1 <i>Progress beyond the state-of-the-art</i>	38
2.4.2 <i>Design methodology</i>	39
2.4.3 <i>Implementation</i>	41
2.4.4 <i>Connection to other components</i>	42
2.4.5 <i>Demonstration</i>	42
2.4.6 <i>Performance</i>	44
3 DATA CORPUS AND AUGMENTATIONS.....	47
3.1 REALISATION OF THE MARVEL DATA CORPUS.....	48
3.2 DATA AUGMENTATION STRATEGY	48
3.3 THE GRAPHICAL USER INTERFACE.....	56
4 POSITIONING WITHIN THE E2F2C CONTINUUM.....	59
4.1 THE DMP TOOLKIT ARCHITECTURAL APPROACH.....	59
4.2 THE MARVDASH FACILITATOR.....	64
5 FUTURE PLANS TOWARDS D2.4 AND RELATED KPIS	69
5.1 DATANA	69
5.2 DFB	70
5.3 STREAMHANDLER.....	70
5.4 HDD	71
5.5 DATA CORPUS AND AUGMENTATIONS	71

5.6 COMPONENT RELATED KPIS 72

6 CONCLUSION 75

7 REFERENCES..... 76

DRAFT

List of Tables

Table 1: DatAna data flows for R1	22
Table 2: Results of the measurements for DatAna component.....	22
Table 3: Results of the measurements for DFB component	30
Table 4: The augmenters that have been used and tested for the augmentation process.....	52
Table 5: Snapshots of the augmented files produced, based on input file from the stage recording of Piazza Maggiore. The produced augmented files have been set to 10, while the Gaussian noise that has been injected is created randomly	53
Table 6: Snapshots of the augmented files produced, based on input file from the stage recording of Piazza Maggiore. The produced augmented files have been set to 5 while the brightness value applied to each produced file has been created randomly.	55
Table 7: NiFi installation with Helm	65
Table 8: Taint nodes command.....	67
Table 9: Snippet of YAML containing tolerations and node affinity	67
Table 10: Updated component KPI status since D1.2.....	72

List of Figures

Figure 1: Data Management and Distribution and Data Corpus-as-a-Service subsystems of the MARVEL architecture (as reported in D1.3).....	12
Figure 2: DatAna topologies	16
Figure 3: Example data flow in NiFi.....	20
Figure 4: DFB overall architecture	27
Figure 5: DFB internal architecture and interactions for MARVEL initial version of integrated framework	28
Figure 6: Indicative screenshot of the DFB monitoring UI dashboard.....	29
Figure 7: StreamHandler – Conceptual Architecture.....	31
Figure 8: StreamHandler - High level Architecture.....	32
Figure 9: Big Data Platform – Monitoring Dashboard	34
Figure 10: StreamHandler AV implementation.....	36
Figure 11: Toy example: Apache Kafka topic partitioning	38
Figure 12: HDD’s BroMin algorithm code.....	42
Figure 13: HDD’s BroMax algorithm code	42
Figure 14: HDD simulation performance	46
Figure 15: The main Data Corpus data flow.....	47
Figure 16: Internal architecture for the core MARVEL Data Corpus repository	48
Figure 17: An example of a video augmentation script. The latter will rotate the input video file towards a randomly generated angle and add a randomly Gaussian noise to the input file. The final output files will be also flipped.....	52
Figure 18: Usage example of the augmentation script in python with the respectively input parameters	52
Figure 19: A snapshot of the augmentation script where the augmentation technique that is used is only the Gaussian noise and with no flipping	53
Figure 20: Corpus GUI – Main Dashboard (mock-up data)	57
Figure 21: Corpus GUI – Add new dataset.....	57
Figure 22: Corpus GUI – Edit dataset.....	58
Figure 23: The DMP in the overall system architecture for use case GRN3: Traffic Conditions and Anomalous Events	61
Figure 24: The DMP in the overall system architecture for use case GRN4: Junction Traffic Trajectory Collection	62
Figure 25: The DMP in the overall system architecture for use case MT1: Monitoring of crowded areas	62
Figure 26: The DMP in the overall system architecture for use case MT3: Monitoring of parking places	63
Figure 27: The DMP in the overall system architecture for use case UNS1: Drone Experiment.....	63
Figure 28: NiFi Cloud Instance available through MARVdash	65
Figure 29: NiFi Cloud instance	66
Figure 30: Instantiate NiFi though MARVdash.....	66
Figure 31: Taints, tolerations, and affinity.....	67
Figure 32: Data Corpus Proxy Logic	68

List of Abbreviations

ACL	Access Control List
AI	Artificial Intelligence
API	Application Programming Interface
AT	Audio Tagging
AV	Audio Visual
AVAD	Audio Visual Anomaly Detection
AVCC	Audio Visual Crowd Counting
CCTV	Closed-circuit television
DFB	Data Fusion Bus
DMP	Data Management Platform
DMT	Data Management Toolkit
DNS	Domain Name System
E2F2C	Edge to Fog to Cloud
EC	European Commission
ELK	Elasticsearch-Logstash-Kibana
ES	Elasticsearch
GUI	Graphical User Interface
HDD	Hierarchical Data Distribution
HDFS	Hadoop distributed file system
HTTPS	Hypertext Transfer Protocol Secure
IEC	International Electrotechnical Commission
IoT	Internet of Things
ISO	International Organisation for Standardisation
JDBC	Java Database Connectivity
JMX	Java Management eXtensions
JSON	JavaScript Object Notation
KPI	Key Performance Indicator
ML	Machine Learning
MQTT	MQ Telemetry Transport
MVP	Minimum Viable Product
NAT	Network Address Translation
RDBMS	Relational Database Management System
REST	Representational State Transfer

RTSP	Real Time Streaming Protocol
S2S	Site-to-Site
SASL	Simple Authentication and Security Layer
SED	Sound Event Detection
SSL	Secure Sockets Layer
TLS	Transport Layer Security
UI	User Interface
URL	Uniform Resource Locator
VAD	Voice Activity Detection
VCC	Visual Crowd Counting
ViAD	Visual Anomaly Detection
VPN	Virtual Private Network
WP	Work Package

DRAFT

Executive Summary

This document provides a description of the current version of the management and distribution toolkit in the MARVEL project. This is the initial version of the document reporting work conducted in the framework of T2.2 *Data management and distribution*, and T2.3 *Incremental scheme: continuous augmentation of the dataset*. The content of the document focuses on presenting the component design and development of the components provided by the involved MARVEL partners within the first 18 months of the project towards achieving the project objectives. The final version of the management and distribution toolkit in the MARVEL project will be documented near the end of the project (M30) in D2.4 and will contain the enriched set of components further improving the set of data-related capabilities in MARVEL. The presented components belong to the (i) *Data Management and Distribution* and (ii) *Data Corpus-as-a-Service* subsystems of the MARVEL architecture (as reported in D1.3), and they mainly target the second objective of WP2, i.e., to *collect and analyse the nature and format of experimental data assets and prepare them for processing in the following WPs*. Work reported in this deliverable contributes to Objective 1 of the project, i.e., to *leverage innovative technologies for data acquisition, management and distribution to develop a privacy-aware engineering solution for revealing valuable and hidden societal knowledge in a smart city environment*.

Some of the components presented in this document were integrated in the MARVEL *Minimum Viable Product* (MVP), as reported in D5.2. The document starts with a general introduction, providing the purpose and scope of this document, the contributions of the work conducted so far in T2.2 and T2.3 to WP2 and to the project objectives, and its relation to other WPs and deliverables. Then, the components proposed by MARVEL partners are described in more detail, followed by a description of how the new components proposed so far meet the Key Performance Indicators (KPIs) related to the data management and distribution in MARVEL. Specifically, the components targeting data management and distribution are described in Section 2 and the components targeting Data Corpus-as-a-service in Section 3. The first part includes the components *DatAna*, *StreamHandler*, *DFB*, and *HDD*. The second part includes the component *Data Corpus*. The respective positioning of the components within the E2F2C continuum is described in Section 4. The document concludes by summarising the work conducted so far and by describing future plans for enriching and further improving the involved components in the remaining period of the project (up to M30) in Section 5. Some of the new methodologies described in this deliverable have been published or are currently under review in international conferences and journals. The corresponding papers or publicly available preprints will be available on the MARVEL website upon publication.

1 Introduction

1.1 Scope and purpose

Deliverable D2.2, entitled “Management and Distribution toolkit – initial version”, reports on the activities carried out within Tasks T2.2 and T2.3, during the period of M06 (June 2021) to M18 (June 2022). The contributions to those Tasks are in the form of components which target addressing data management, storage, and distribution in the context of smart cities. The components aim at improving performance and/or efficiency in tasks involving data flows, towards achieving the objectives of the project, and specifically Objective 1, i.e., to leverage innovative technologies for data acquisition, management and distribution to develop a privacy-aware engineering solution for revealing valuable and hidden societal knowledge in a smart city environment. This will be implemented by achieving the second objective of WP2, i.e., to collect and analyse the nature and format of experimental data assets and prepare them for processing in the following WPs. The report provides a detailed description of the components proposed by the project partners, considering the restrictions set by the application scenarios in MARVEL, such as the solution scalability, data nature and availability, as well as application requirements and constraints. The components described in this deliverable will be enriched by work conducted in the remaining period of the project (up to M30), which will be reported in D2.4.

1.2 Contribution to WP2 and project objectives

The work conducted so far in T2.2 and T2.3 and reported in this deliverable contributes to the second objective of WP2 “MARVEL multimodal Data Corpus-as-a-Service for smart cities”, i.e., to collect and analyse the nature and format of experimental data assets and prepare them for processing in the following WPs. Components in this task aim at addressing limitations of existing solutions, and providing state-of-the-art capabilities to the Data Management and Distribution and Data Corpus-as-a-Service subsystems of the MARVEL architecture. The subsystems include functionalities which are implemented in five components:

1. *DatAna*, an Apache NiFi and FIWARE-compliant tool for data flows definition with an easy-to-use user interface for developers of big data solutions.
2. Data Fusion Bus (*DFB*), a customisable trustworthy tool for transferring large volumes of heterogeneous data between several connected components and the permanent storage.
3. *StreamHandler*, a high-performance, distributed streaming tool for handling real-time data.
4. Hierarchical Data Distribution (*HDD*), a smart data distribution methodology for Apache Kafka topic partitioning under heterogeneous parameters, optimising the involved use of resources.
5. *Data Corpus*, a Big Data repository handling datasets from smart city environments and potentially sharing it with research and industrial communities.

Work performed during the first 18 months of the project proposed components for improving the data flow, distribution, management and storing across the Edge-to-Fog-to-Cloud (*E2F2C*) continuum, taking into account the pilot requirements, constraints, and available data. Work targeting further automating and improving the efficiency of the current management solutions has been planned to be conducted during the remaining part of the project, establishing a

continuous development and integration process, in collaboration also with the activities of WP5 as well as the pilot developments.

Figure 1 illustrates the (i) *Data Management and Distribution* and (ii) *Data Corpus-as-a-Service* subsystems of the MARVEL architecture (as reported in D1.3 [1]). The subsystems, using the above-described components provide the initial functionality of the data management and distribution process in MARVEL. As such, the new components developed within T2.2 and T2.3 primarily contribute to Objective 1 of MARVEL, i.e., to leverage innovative technologies for data acquisition, management, and distribution to develop a privacy-aware engineering solution for revealing valuable and hidden societal knowledge in a smart city environment.

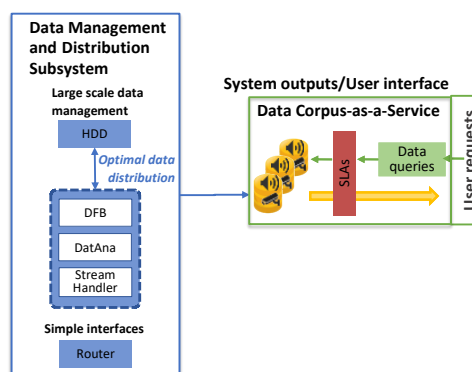


Figure 1: Data Management and Distribution and Data Corpus-as-a-Service subsystems of the MARVEL architecture (as reported in D1.3)

1.3 Relation to other WPs, deliverables, and activities

Within WP2, work conducted in T2.2 and T2.3 will be connected to work in T2.4. *Sharing multimodal Corpus-as-a-Service: fostering the European data economy vision in smart cities*, which intends to make possible for MARVEL to share its own corpus in a secure environment where innovative applications can be built by third parties. In this respect, the components of T2.2 and T2.3 will be used to maximise the impact that the MARVEL corpus will have onto the international scientific and research community when it comes to data management and distribution for smart cities.

Work in T2.2 and T2.3 is also related to multiple tasks belonging to other work packages. Specifically, it is related to T1.3 *Experimental protocol – real life societal trial cases in smart cities environments*, as reported in D1.2. The management and distribution functionalities in T2.2 and T2.3 and the respective data-oriented components are described in T1.4 *Technology convergence: specifications and E2F2C distributed architecture*, which refines the specification of the conceptual architecture of the MARVEL E2F2C ubiquitous computing framework as reported in D1.3. The AI models in T3.3 *Multimodal audio-visual intelligence* will be trained on the Data Corpus of T2.3, which will be collected, analysed, managed, and distributed according to WP2 MARVEL’s multimodal Data Corpus-as-a-Service for smart cities, and will then be fed into T4.4 *MARVEL’s decision-making toolkit* implementing the interactive visualisation and audio-visual analytics tools. The management and distribution functionalities developed in T2.2 and T2.3 through the components mentioned above are continuously being integrated in the MARVEL architecture, WP5 tasks T5.1 *HPC infrastructure*, T5.2 *Resource management and optimised automatic usage*, T5.3 *Continuous integration towards MARVEL’s framework realisation*. The effectiveness of the provided components will be evaluated in T5.4 *Quantifiable progress against societal, academic and*

industrial validated benchmarks and in T5.5 *From the prototype to the final solution*. Three components (DatAna, DFB and Data Corpus) were included in the MARVEL Minimum Viable Product (MVP) as reported in D5.1 [2]. Five components (DatAna, DFB, StreamHandler, HDD and Data Corpus) were included in the initial release of the MARVEL integrated framework (D5.4). Finally, the work in T2.2 and T2.3 is connected to WP6 *Real-life societal experiments in smart cities environment*, where the above-mentioned data management and distribution components will be used for decision-making in real smart city environments.

DRAFT

2 Components of the DMP toolkit

The DMP toolkit consists of the following components:

- DatAna is based on the Apache NiFi ecosystem of tools (mainly NiFi, MiNiFi and NiFi Registry). The tool relies on the definition of data processing flows using the NiFi user interface. NiFi provides an extensible ample set of off-the-shelf data processors allowing ETL features, as well as the possibility of connecting different layers using the NiFi Site-to-Site (S2S) protocol, of special interest for MARVEL as it allows communication and movement of data in the computing continuum. The main role of DatAna is in the data inference pipeline, collecting the results from the inference models, transforming and aligning them to the expected MARVEL data models (based on FIWARE Smart Data Models) and passing the results to the DFB for further processing and storage.
- DFB is a customisable component that implements a trustworthy way of transferring large volumes of heterogeneous data between several connected components and the permanent storage. It comprises a collection of dockerised, open-source components which allow easy deployment and configuration as needed. DFB's architectural design addresses several challenges that are raised by both the large volume and the heterogeneous nature of data from different sources, taking into consideration the needs and restrictions of the employed components. The main addressed challenges include seamless aggregation of data with different formats, and access of data through a common, safe, accessible interface.
- StreamHandler is a fully-featured industrial-grade solution that: i) is capable of scaling out and accommodating various and from different domains big data, interoperating with all modern data storage technologies as well as other persistence approaches and ii) can support important Big Data languages including Python, Java, R and Scala as well. In its current form, StreamHandler delivers a high-performance (low latency and high throughput) distributed streaming option for handling real-time data based on Apache Kafka. It can efficiently ingest and handle massive amounts of data into processing pipelines, for both real-time and batch processing. The platform and its underlying technologies support of data-intensive ICT services from cloud to edge. In the context of MARVEL, StreamHandler functionality is extended for operation with AV data, allowing an efficient real-time AV stream segmentation for storage and retrieval.
- HDD targets smart data distribution in networked environments with heterogeneous parameters, optimising the involved use of resources. In its current form, HDD considers the problem of Apache Kafka data topic partitioning optimisation [3]. Even though Apache Kafka provides some out-of-the-box optimisations, it does not strictly define how each topic shall be efficiently distributed into partitions. HDD models the Apache Kafka topic partitioning process for a given topic. Then, given the set of brokers, constraints and application requirements, HDD formulates the optimisation problem of finding how many partitions are needed. Furthermore, HDD implements two simple, yet efficient heuristics to solve the problem.

The components are presented in detail in the following subsections.

2.1 DatAna

DatAna is a component for streaming and batch data acquisition and transformation based on the Apache NiFi¹ ecosystem of tools. As one of the DMP components of MARVEL, it plays a central role in the distribution and management of data among the layers of the computing continuum. DatAna's main role is exerted in the real-time inference pipelines, gathering the results of the inference models, transforming and driving them to the cloud and to the DFB for further processing and storage.

2.1.1 Progress beyond the state-of-the-art

DatAna's role is to provide a set of artifacts to allow data manipulation close to the source. As such, the component can be deployed in the different layers (edge, fog, and cloud) to be able to get the data from the inference models and perform the necessary transformations and enrichment to make the results available for the rest of the layers in a seamless way. This is achieved by defining NiFi topologies between cloud, fog, and edge to enable the movement of the data in a secured way.

DatAna provides a powerful set of off-the-shelf data processors included in NiFi for acquiring, processing, and outputting data. The data management pipelines can be easily drawn in the NiFi node-based graphical user interface by dragging and dropping data processors and connections, with low or even no code effort. Therefore, DatAna helps to easily develop and maintain otherwise tedious data transformation jobs, thus reducing the complexity of implementing inference pipelines, while ensuring the scalability and traceability of the solution. DatAna can be scaled as required (horizontal and vertical scalability) to increase the throughput of the system if necessary. DatAna inherits other NiFi properties such as guaranteed delivery, tolerance to loss (queueing of data), back pressure (how much data can be queued in a connection to avoid data overflow), dynamic prioritisation, control of the data flow (warm deployment, start and stop at any time), and security in the connections for data transmission and access.

2.1.2 Design methodology

NiFi essentials. DatAna relies in the Apache NiFi ecosystem to perform data management in the E2F2C continuum. Before entering into the details on how this is achieved, let's have a look at the main concepts related to NiFi.

NiFi is a data processing engine, and as such its main goal is to gather, transform and enrich a piece of data from a start of a data flow to its end. Therefore, the first main concept in NiFi is that of a **flowfile**, which is the data payload and a set of metadata (attributes) associated to this payload. A flowfile passes through a sequence of **processors**, self-contained components that perform the necessary data operations. There are hundreds of off-the-shelf NiFi processors for acquiring, routing, transforming and outputting data to databases or other systems. The processors are pipelined using **connections** and can be dragged and dropped in the **NiFi UI**, a powerful tool to design the data flows. Most of the work done in NiFi is related to the design of data flows for specific data in the UI by pipelining and configuring processors, using the NiFi Expression language or other processors for data transformation and connecting to external systems, with zero or low coding effort.

DatAna in MARVEL. DatAna comprises the following elements:

¹ <https://nifi.apache.org/>

- One instance of the Apache NiFi in the MARVEL cloud and one instance in each of the fog servers from the pilots.
- Apache MiNiFi² agents in edge infrastructure.
- Apache NiFi Registry³ for data flows version control sitting on the MARVEL cloud.
- Mosquitto MQTT⁴ message broker as a supporting messaging tool to communicate with the inference models. Structurally speaking, MQTT is not part of DatAna. However, in MARVEL it has been decided to include the MQTT broker to decouple the messaging among components and DatAna. Therefore, an MQTT broker has been released with DatAna in a separate docker container to be able to manage those interactions in the different layers. In principle, each deployment of DatAna in MARVEL will have an associated MQTT deployment in the same layer to be able to capture the results of the inference in the layer where DatAna is operating. Specific MQTT topics are set for each inference model result, similar to what is done for the DFB in Kafka.

DatAna makes use of these NiFi ecosystem tools and instances to enable the creation of specific secured NiFi topologies to process, transform, enrich and route the data from its source to the required destination system. This is enabled via de NiFi Site-to-Site (S2S)⁵ communication protocol. S2S allows an easy transfer of data from one NiFi or MiNiFi instance to another in a secured way (socket-based protocol and HTTPS are supported). Figure 2 shows an example of the DatAna topologies for MARVEL, where the communication among the different layers (edge, fog, and cloud) is depicted via arrows enabled by the S2S protocol, the connection to the inference models via MQTT brokers. The output of DatAna will be provided in the cloud layer to dedicated Kafka topics in the DFB.

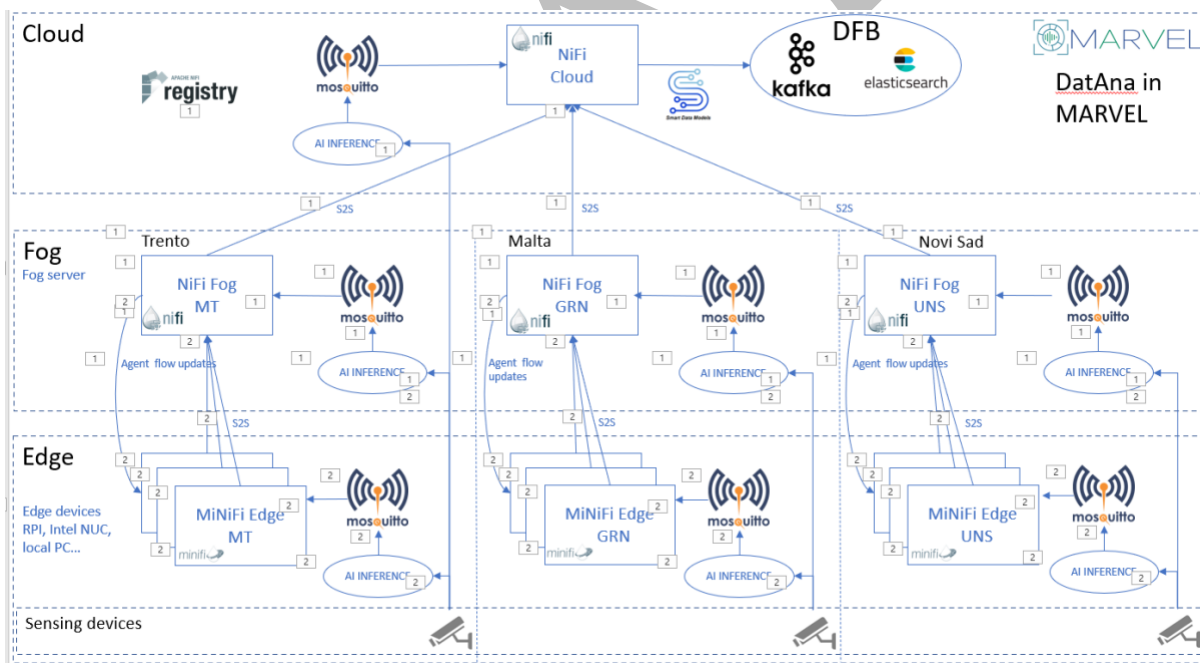


Figure 2: DatAna topologies

² <https://nifi.apache.org/minifi/>

³ <https://nifi.apache.org/registry.html>

⁴ <https://mosquitto.org/>

⁵ https://nifi.apache.org/docs/nifi-docs/html/administration-guide.html#site_to_site_properties

Best practices for configuration. Before starting to use NiFi, it is important to configure it properly. Some of the best practices for configurations are the following:

- Usage of docker container for the deployment of NiFi. Besides the mandatory usage of docker in MARVEL to be able to function in a Kubernetes environment, it is a good practice to use NiFi docker distribution. However, it is recommended to separate internal storage (repositories and logs) to the docker distribution to enable handling of these repositories separated from the docker. This would allow for instance an easy version upgrade of NiFi, as the internal repositories would likely not be affected by the new version.
- Use separate disks for the internal NiFi repositories. It is a good practice to use different disks, especially for the internal provenance and content repositories due to the high usage of the disk. It is also advisable to use different disks of other high I/O disc access systems, such as Kafka, which might reside in the same infrastructure.
- Use secure NiFi communications. Since v1.14.0, NiFi comes with built-in security by default. In MARVEL, we opted for using a combination of user credentials and TLS security, using keystores, truststores and appropriate certificates. This enables the possibility of communicating data among the E2F2C layers in a secured way.

The usage of NiFi Registry buckets and reusable data flows. The usage of NiFi Registry⁶ for versioning of specific data flows can help two main purposes: 1) keep track of the versioning of the data flows as they evolve; and 2) reuse the data flows in different NiFi instances when appropriate. As the data flows of specific inference models can be reused in different NiFi instances (i.e., a data flow for handling the outputs of the SED can be used at the fog or cloud layers), to avoid code and flow replication it is advisable to save and use NiFi Process Groups stored in the NiFi Registry containing the common parts to be reused. The creation of specific process groups containing specific reusable data flows blocks in the form of process groups saved in the NiFi Registry is one of the best practices followed in MARVEL.

Best practices for designing good data flows. There are several best practices adopted by the NiFi community^{7,8} in order to make a good design of the data flows. In MARVEL, we recommend the following best practices:

- Flow layout in the NiFi UI: It is important to follow some specific conventions when designing data flows in order to make it more maintainable and easier to understand by other developers:
 - Flow direction: Aligning the data flow from top to bottom following a “gravity” principle (gravity pulls down objects on Earth, and similarly data in the flow goes down as it is further processed). On the other hand, conditionals and error handling should be drawn horizontally. It is possible to select several processors and with the right click select “align vertically” in the NiFi UI to facilitate this process.
 - Put descriptive names to the processors that explain what they do instead of keeping the default names of the processors given by NiFi.
 - Add documentation as comments to the processors explaining the operation they do (comments appear as a triangle in the processor right-bottom corner and visualise on mouse over).

⁶ <https://nifi.apache.org/docs/nifi-registry-docs/index.html>

⁷ <https://benyaakobi.medium.com/design-patterns-in-nifi-f9ad1cb02588>

⁸ <https://www.youtube.com/watch?v=RjWstt7nRVY>

- Drag labels into the flow to explain what the specific area of the flow is doing, but do not overdo this if it is self-explanatory.
- Do not overuse the primary node: When the first processor only runs in the primary node (i.e., reading files from a Kubernetes private or shared folder), it is clear that in the scheduling tab of the properties of the processor, the execution should be set to the primary node. However, the rest of the processors in the data flow shouldn't be set to the primary node. This node may shift, and the data are queued. There is no single use case to set the next processors to primary node (unbalanced). Use “single node” as a load balancing strategy.
- Be careful with load balancing: Load balancing data means taking data from a node of a NiFi cluster and moving around to several nodes. In NiFi this is not done automatically but managed in the data flow. It is therefore dependent on the cluster nodes available and requires drawing the line between the efficiency gain of processing the data across the cluster vs. the cost of moving the data. To select the load balancing strategy, you should click on the connection between 2 processors and in the settings select your strategy (i.e., round robin – evenly distribute the data).
 - If the data comes from a bucket with no queuing semantics, we should always get the data in only one node (first Processor that fetches the data). In fact, if we would like to mark that the data comes from all nodes, we would be fetching the same data from all nodes (which would be highly inefficient).
 - Once the data has been load-balanced in the first node, the data is already balanced. You shouldn't again balance for the rest of the nodes, or you would be moving data around all the time.
 - If the data comes from a node with queuing semantics (i.e., Kafka and most of the pub-sub or similar queuing systems), you don't need to set load balancing. Use NO LOAD BALANCING INSTEAD.
- Select the right processor for the task to be performed over the data: NiFi provides hundreds of processors. Some of them can be used in combination to perform data transformations and enrichment. Try to look at different possibilities and select the most appropriate one:
 - For instance, record-oriented processors in NiFi (i.e., ConsumeKafkaRecord) are typically more performant to fetch and make format transformations of records than other processors and provide record readers and writers. Do not use these processors in combination with others to perform data transformations (i.e., ConvertRecord after ConsumeKafkaRecord), as the first processor could actually perform direct data transformations.
- NiFi provides a complex expression language to perform data transformations: Do not overuse it if there are other possibilities, as it takes processing time. There are ways of implementing more performant expression language formulas. Check carefully the NiFi help for expression language.

2.1.3 Implementation

Dockers and configuration. In order to implement DatAna, several instances of NiFi have to be deployed in the different pilot layers and infrastructure and configured to enable the communication among the required taxonomy of instances.

We have provided the NiFi and MiNiFi dockers (v1.15.3, the last version used as per beginning of March 2022), as well as uploaded one version of a Mosquitto MQTT broker to MARVDash

as required. In the case of NiFi, a Helm Chart for NiFi based on the cetic helm chart⁹ has been provided to FORTH to be tailored for MARVdash deployment. As a result, NiFi is ready to be deployed as a service in MARVdash. To enable secure communication, the different NiFi instances must be protected using TLS security by populating the necessary keystores and trust stores, as well as enabling the security in NiFi. All the instructions to enable the security and the configuration have been provided in the MARVEL GitHub for DatAna¹⁰, and will be made available publicly.

Current set of DatAna instances.

- NiFi Registry
- NiFi Cloud
- NiFi Fog
 - MT
 - GRN
 - UNS
- MiNiFi Edge
 - MT
 - GRN
 - UNS

Data Flows. Several data flows to gather the outputs from the difference inference components have been prepared. Figure 3 shows a screenshot from the definition of one of these data flows in the NiFi UI as an example of the type of flows implemented:

⁹ <https://github.com/cetic/helm-nifi>

¹⁰ <https://git.marvel-project.eu/marvel/datana/datana/-/tree/main/docker/secureNifi>

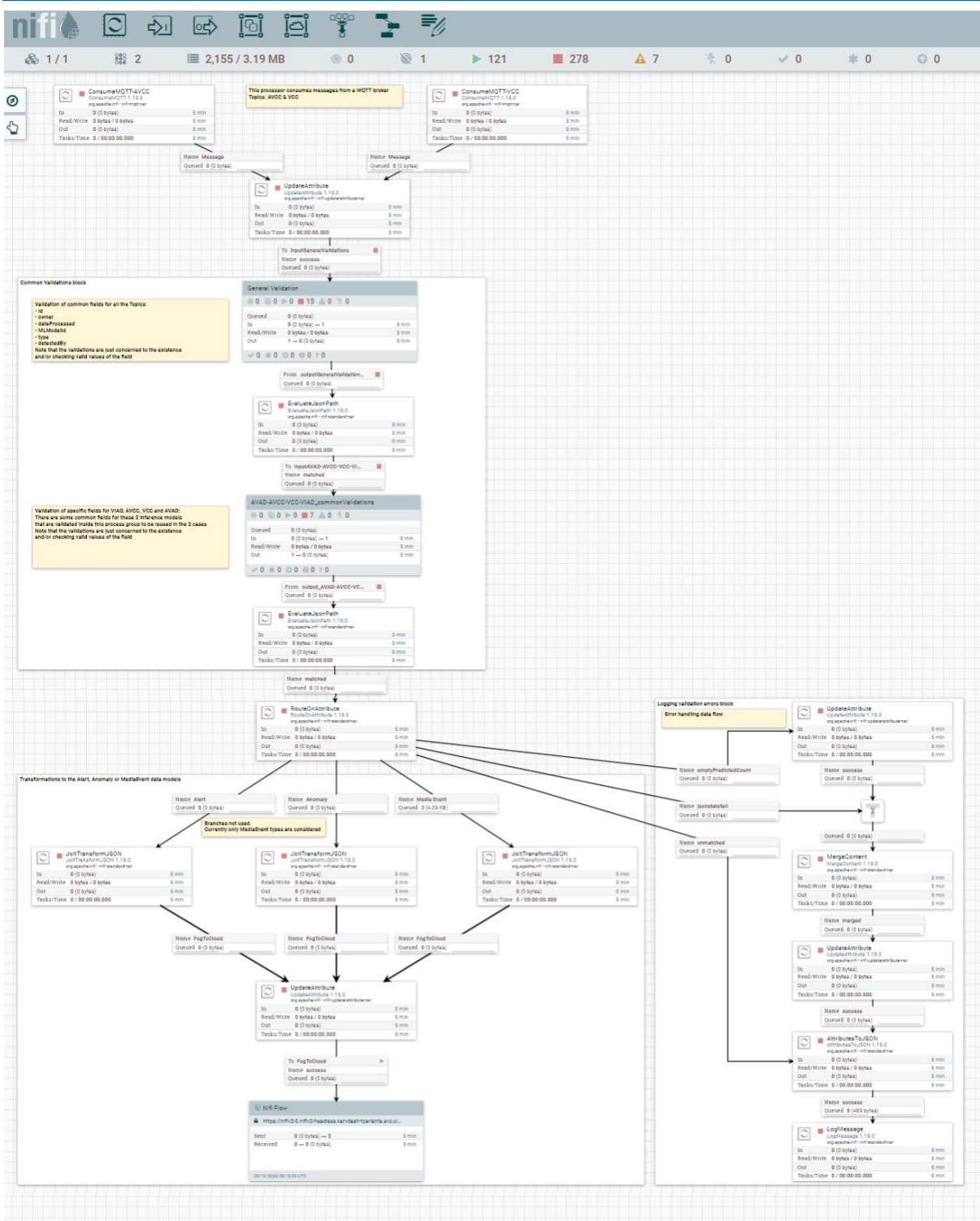


Figure 3: Example data flow in NiFi

This data flow, corresponding to the handling of the outputs of the Audio-Visual Crowd Counting (AVCC) and Visual Crowd Counting (VCC) inference models, shows several NiFi processors and connections in action since the gathering of the data from the MQTT broker (from topics corresponding to the abovementioned inference models) to the error handling, transformation and enrichment of the data to match one of the data models for Alert, Anomaly or MediaEvent. The output of the data is then directed to the last step of the pipeline, which

could be either an output port to connect to another NiFi instance in the topology (e.g., if the processing is done in the edge or fog layers, as in the example shown in the figure), or to input the data to the Kafka topic of the DFB in the cloud layer.

2.1.4 Connection to other components

DatAna connects to all model inference components gathering their outputs via dedicated topics in MQTT message brokers. These topics are named after the acronym of the inference model (CATFlow, TAD, ViAD, AVAD, VCC, AVCC, SED, AT, or VAD).

Dedicated output data models for each of the inference models results have been defined for each of them. These data models contain some specific metadata fields that are expected to be populated by the inference models to DatAna, such as timestamps (either a single time, or a start and end time of the inferred event), the id of the event, the model id of the inference model that produced the results, the camera id of the device (camera, microphone or component origin of the AV stream), owner (the MARVEL pilot use case name -i.e., “GRN3”- to which the inference is applied) or the detectedBy or device id in which the inference processing takes place. After transforming the outputs to the MARVEL data models (Alert, Anomaly or MediaEvent), the results are finally provided either to another NiFi upper in the topology managed by DatAna, or to dedicated Kafka topics in the DFB.

2.1.5 Demonstration

As mentioned in the previous sections, DatAna has been deployed in several layers of the infrastructure and comprises several tools, such as Apache Registry at the cloud for data flow versioning control, Apache NiFi in the cloud and fog environments and MiNiFi at the edge. As commented before, the taxonomy of components of DatAna requires extra configurations to enable NiFi S2S communication between MiNiFi (edge), NiFi (fog) and NiFi (cloud).

All these NiFi elements have been deployed using MARVdash guidelines. In particular, we used the Apache NiFi and MiNiFi docker image v1.15.3 from the docker hub^{11, 12} in combination with a helm chart of NiFi based on the cetic helm chart as described in previous subsections. For ARM-based systems (i.e., Raspberry Pi) a dedicated docker image of MiNiFi has been uploaded to MARVdash. Specific YAML files for the deployment of NiFi and MiNiFi in MARVdash have been also produced and uploaded, including some properties to be populated when configuring each instance to indicate the deployment infrastructure, location of the internal NiFi repositories, etc.

Once deployed, each of the instances of NiFi/MiNiFi must be connected to create the taxonomy of DatAna depicted in Figure 2. To achieve this, the NiFi S2S protocol properties have to be configured using security TLS certificates. This means that after the deployment of all instances as MARVdash services, some updates of the services configurations are required to copy and make available the adequate certificates pointing to the specific end points where the NiFis in the different layers have been deployed.

DatAna does not include a message broker as part of the component. However, to decouple and make an easier integration of the results of the inference models with DatAna, the inclusion of a message broker (MQTT) has been proposed and decided to enable the inference results to

¹¹ <https://hub.docker.com/r/apache/nifi>

¹² <https://hub.docker.com/r/apache/nifi-minifi>

reach DatAna in a streaming fashion. To enable this deployment of message brokers at the different layers using MARVDash, docker images of Mosquitto MQTT¹³ have been provided.

A set of data flows have been defined for each of the inference model output ingestion, transformation and compliance with the data models required by the DFB. These data flows have been prepared in the required DatAna instances and layers in accordance with the use cases implemented in the pilots for M18. Table 1 reports the list of data flows implemented for R1 by use case/layer. This is a list showcasing the components in each of the layers for which specific data flows have been defined in the NiFi UI:

Table 1: DatAna data flows for R1

Layer	Use cases				
	GRN3: Traffic Anomalous Events	GRN4: Trajectories	MT1: Crowd Monitoring	MT3: Parking Lot	UNS1: Drone Crowd Classification
Cloud	AVAD	AVCC	ViAD		
	AT	SED	VCC	SED	
				AT	
				AVAD	
Fog	CATFLOW	CATFLOW	CATFLOW		AVCC
	TAD	TAD	TAD		
Edge	CATFLOW	CATFLOW		VAD	VAD
	TAD	TAD			

2.1.6 Performance

DatAna is a data management platform sitting in the heart of the data inference pipeline by acquiring the outputs of the inference models, transforming them into the required smart data models and routing the data from the edge to the fog and cloud towards the DFB. Therefore, the main metrics related to the work of DatAna are throughput and latency.

During the tests performed within the MVP, the performance metrics of a single NiFi instance were measured. Table 3 summarises the collected measurements for the specified metrics.

Table 2: Results of the measurements for DatAna component

Metric	Value
Data loss rate	0
Service availability-failed request	100% availability
Data access restriction	None
Data throughput	Scenario 1: 0.54 MB/s (25.6 KB for 234 Kafka entries) Scenario 2: 1.1 MB/s (11 files, total of 76.6KB from SED data, corresponding to 1731 Kafka entries)
Response time	Scenario 1: 47.1 ms

¹³ https://hub.docker.com/_/eclipse-mosquitto

	Scenario 2: 67 ms
Number of cluster nodes	1

For a more in-depth analysis of performance metrics, there is this Cloudera study¹⁴, which reports how NiFi behaves in terms of scalability and performance (data rates) using very demanding workloads.

DRAFT

¹⁴ <https://blog.cloudera.com/benchmarking-nifi-performance-and-scalability/>

2.2 DFB

The Data Fusion Bus (DFB) is a customisable component that implements a trustworthy way of transferring large volumes of heterogeneous data between several connected components and the permanent storage. It comprises a collection of dockerised, open-source components which allow easy deployment and configuration as needed.

DFB's architectural design addresses several challenges that are raised by both the large volume and the heterogeneous nature of data from different sources, taking into consideration the needs and restrictions of the employed components. The main addressed challenges include:

- seamless aggregation of data with different structures or formats;
- a cluttering threat to the components due to the quantity of the input data;
- access of data through a common, safe, accessible interface.

Inherent to DFBs design is the efficient handling of the enormous volume of the data that need storage and manipulation, as well as mechanisms to remediate potential bottlenecks, lag, or high demand on network traffic. These design decisions enable horizontal scalability while providing a solution that is cloud-native with stateless components capable of being deployed with flexibility. DFB follows the middleware approach by aligning data streams for time and granularity and creating a user interface that serves as the interface of the platform, customised to aggregate multiple streams, thereby allowing seamless service of data to the network analysis and visualisation.

2.2.1 Progress beyond the state-of-the-art

The DFB is based on state-of-the-art industrial-grade technologies and is optimised for IoT applications, especially in the context of smart cities.

A key technology of the DFB is the Apache Kafka message broker technology and its associated extensions¹⁵ [4] [5]. Apache Kafka is currently an industry standard for distributed messaging systems, providing a wide range of advantageous features, i.e., powerful event streaming, fault-tolerance and reliability, high scalability, open-source status, multi-tenancy, real-time processing, high suitability for big data management and supported by a large community and associated ecosystem. Apache Kafka is used for "building real-time streaming data pipelines that reliably get data between systems or applications" supporting multiple sources that can even be in a distributed system. As a message broker, it offers higher fault tolerance in comparison to traditional message brokers and it can support significantly higher throughput. It is thus capable of handling great amounts of data and deliver them in real-time to the components that require them. It can support multiple topics, which are used by producers and consumers of data to write and read data from respectively. A new topic can be easily created when needed in order to support the communication between new components or the new component can become a publisher or subscriber of an already existing topic. Multiple producers can write messages on the same topic and multiple subscribers can receive messages from the same topic. Subscribers can either compete for the same message so that only one of them will receive it if they belong to the same group or they can all receive it if it is needed by multiple components. Apache Kafka offers various options for security as well, which include encryption of the data and client authentication and authorisation, allowing only specific publishers and subscribers per topic. Kafka can be deployed in a cluster comprised of brokers located in the edge or fog layers and a (central) broker in the cloud installation. Data Replication over secure channels between the cluster nodes provides scalability, reliability, and security.

¹⁵ <https://www.youtube.com/watch?v=tseAvqK8JEA>

Another key technology implemented in the DFB is the Elasticsearch data storage technology [6], which has also matured to become an industry standard for a wide range of applications. Elasticsearch offers high scalability and interoperability, fast performance, versatile and powerful search engine, API-driven and schema-free design, multi-tenancy, open-source status and orientation towards documents. Elasticsearch is a distributed RESTful search engine built for the cloud. It resembles a NoSQL database being able to store data as JSON documents but also works as a powerful search engine. Data is stored in the indices, which can be separated into shards and stored in a distributed system with replicas if needed. In contrast to traditional databases, the stored documents can have a different format. Within MARVEL, data collected in the central Kafka broker is persisted in the Elasticsearch cluster. The Elasticsearch cluster can also be used by other MARVEL components for querying historical data.

The DFB also integrates Keycloak¹⁶, which is an open-source software product to allow single sign-on with Identity Management and Access Management aimed at modern applications and services. Within MARVEL, it can be used for securing access to the DFB Core and UI (and possibly to other components).

Beyond the seamless integration of the above technologies, the DFB offers a novel tool for the central management and monitoring of the DFB operation. The DFB core is a Springboot application that exposes a REST API for managing and monitoring the DFB components (Kafka cluster and Elasticsearch cluster). The DFB UI provides a GUI that acts as a client to the DFB Core. The key features are:

- Health monitoring (Kafka broker status, ES cluster status, replication status)
- Kafka Performance metrics (e.g., broker status, messages in/out per topic, bytes in/out per topic, consumer lag)
- Elasticsearch Usage metrics
- Kafka Topic management (creation, configuration, persistence by Elasticsearch, client authorisation & authentication)

The aforementioned technologies that are integrated in the DFB enable the following key capabilities:

- Data aggregation from heterogeneous data sources and data stores.
- Real-time analytics, offering ready-to-use ML algorithms for classification, clustering, regression, and anomaly detection.
- An extendable and highly customisable User Interface for Data Analytics, manipulation, and filtering, as well as functionality for managing the platform.
- Web Services for exploiting the platform outputs for Decision Support.
- Applications for Smart Production, Digitisation, and IoT, among others.

In the context of MARVEL, the DFB is configured to fulfil the following objectives:

- Aggregate inference results originating from all AI components using FIWARE-compliant data models.
- Re-distribute incoming inference results in real-time to SmartViz for presentation to the user and to the Data Corpus for AI training purposes.
- Provide access to historical inference result data to SmartViz for presentation to the user.

¹⁶ <https://www.keycloak.org>

- Accept inference result verification data from users accessing SmartViz to update the respective historical data entries accordingly and relay them to the Data Corpus to improve the body of labelled data for AI training purposes.
- Perform data fusion operations for inference results of selected AI components.
- Provide performance metrics related to the DFB operation, visualised through an intuitive UI.

These objectives and the associated functionalities of the DFB complement the role of the other DMP components (DatAna, StreamHandler) to provide significant added value to the smart city use cases addressed by MARVEL.

Due to the inherent high scalability of the DFB and its potential distribution across infrastructure nodes, it can accommodate increased throughput and latency demands, when provided with sufficient infrastructure computational resources.

Nevertheless, in the context of MARVEL, further potential optimisations in the performance of the DFB are investigated through its interaction with the HDD component. These investigations concern the configuration and operation of the Apache Kafka message brokers, which are considered to be at the core of the DFB technologies. More specifically, HDD will seek to provide optimisation recommendations for the configuration of the Kafka topic partitions¹⁷, which has a pivotal role in the overall Kafka messaging system performance. To that end, the DFB is configured to provide the Kafka topic partitioning information to HDD upon demand along with performance metrics that are associated with the particular partition configuration. The DFB can also accept updated partitioning information from HDD, which is the result of the HDD's internal optimisation engine and apply it in the DFB configuration for subsequent operation.

2.2.2 Design methodology

In the context of the MARVEL MVP, the DFB contributed to the 3 use cases that were defined for the MVP as the engine that fuses and indexes large volumes of heterogenous non-AV data. It served as an interface point for transferring non-AV data collected by the DMP platform, especially DatAna, to the SmartViz's UI, as well as with connections to the MARVEL Data Corpus metadata. In all scenarios, the input to DFB is non-AV data collected and processed by DatAna. The latter component acts as a producer to DFB's Kafka entry point, by publishing to specific topics that are defined per scenario. The data streamed through Kafka are indexed in DFB's Elasticsearch, using DFB's ES-connector. The data streamed through or indexed at the DFB were readily available for any client component, most importantly SmartViz. This user-facing component could access data in two ways: a) by subscribing to the corresponding topic of DFB's Kafka, for real-time visualisations, or b) by querying the DFB's Elasticsearch API, for visualisations of historical data.

Following the release of the MARVEL MVP, the DFB was designed to accommodate the needs of the 5 use cases that were defined for implementation in the context of the initial version of the MARVEL Integrated framework. The functionalities that had been developed for the MVP were further elaborated, refined and extended based on the experience from the MVP and the updated requirements for the initial version of the MARVEL Integrated framework. In this context, the following revisions were made:

¹⁷ https://kafka.apache.org/documentation/#intro_concepts_and_terms

- Configuration for receiving inference results in a FIWARE-compliant data models (MediaEvent, Alert and Anomaly) from DatAna.
- Configuration of an individual Kafka pub/subtopic for each inference result type according to the AI producer component.
- Implementation of inference result verification process, i.e., accept inference result verification messages from SmartViz on a dedicated Kafka topic and update corresponding entries in the Elasticsearch.
- Implementation of a data fusion process for inference results that refer to single instants in time. Consecutive repetitive incoming results from the same AI producer (e.g., anomaly detection) that refer to a single time instant are identified and fused together to form a composite result that refers to a period in time. The start of the time period corresponds to the first occurrence of a repetitive event and the end of the time period corresponds to the last occurrence of that event.
- Revisions of the Elasticsearch API based on updated requirements from SmartViz.
- Implementation of the DFB monitoring stack.

2.2.3 Implementation

The key modules of DFB are:

- Apache Kafka, an open-source framework for stream processing.
- Elasticsearch, a distributed, multitenant-capable, full-text search engine.
- DFB Core & UI, implementation of a REST API and a client GUI, respectively, for management and monitoring of the DFB components.
- Keycloak, an open-source software product that provides single sign-on to applications and services.

Figure 4 depicts the DFB's overall inner architecture and its relation to other components of the MARVEL platform. This figure shows DFB's main modules mentioned above, as well as its interfacing with edge devices that may produce non-binary data and other fog/cloud components that provide their processed data into DFB.

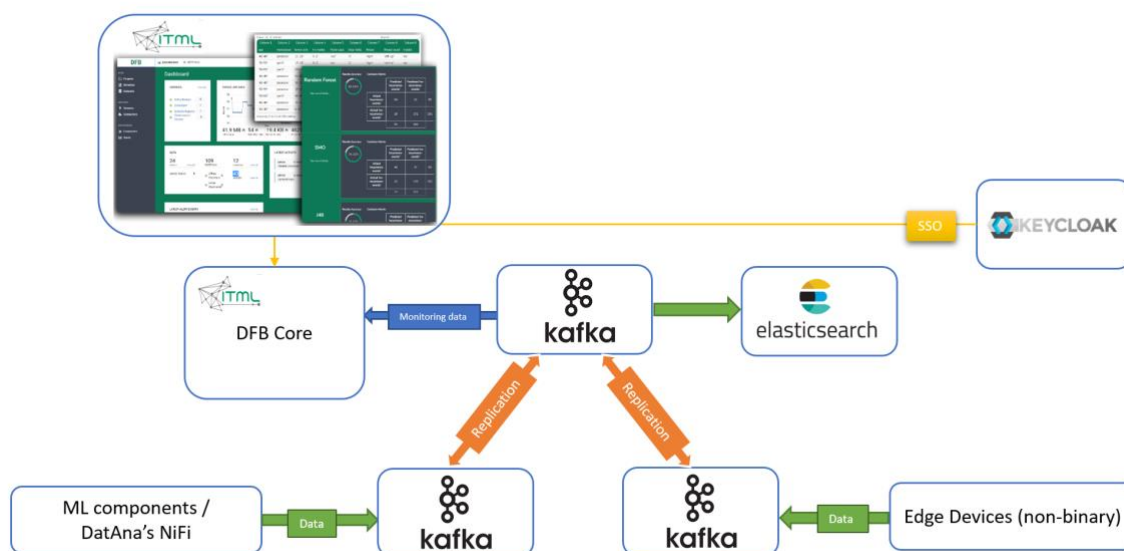


Figure 4: DFB overall architecture

The main inbound interface for DFB is Kafka's messaging system that is based on the publish-subscribe pattern. More specifically, any MARVEL data-producing component, including real-time event detection and data analytics components, connects to DFB and publishes any relevant data to a specific, predefined topic.

Regarding outbound interfaces, in the general case, any data-consuming component can subscribe to a topic and receive instant updates on published data. Within the context of MARVEL, data collected from Kafka brokers is subsequently passed onto an Elasticsearch Logstash Kibana (ELK) stack for storage and further processing and visualisation. Although DFB offers its own graphical UI for the visualisation of aggregated data, collected streamed and stored data can be made available to MARVEL's visualisation components (e.g., SmartViz).

Finally, DFB offers a complete, standalone Single Sign-on module based on Keycloak open-source product to ensure authenticated and authorised access to fused data.

The DFB does not directly process or handle AV data; instead, any results of AV processing from ML components can be made immediately available for real-time analytics or stored for later filtering, processing, searching, and visualisation.

As DFB is typically deployed on the cloud, it does not offer any edge processing options. It is designed and optimised for handling non-binary data that is streamed to the Kafka interfaces in real-time.

Figure 5 illustrates the specific implementation of DFB that was performed for the needs of the initial version of the MARVEL integrated framework.

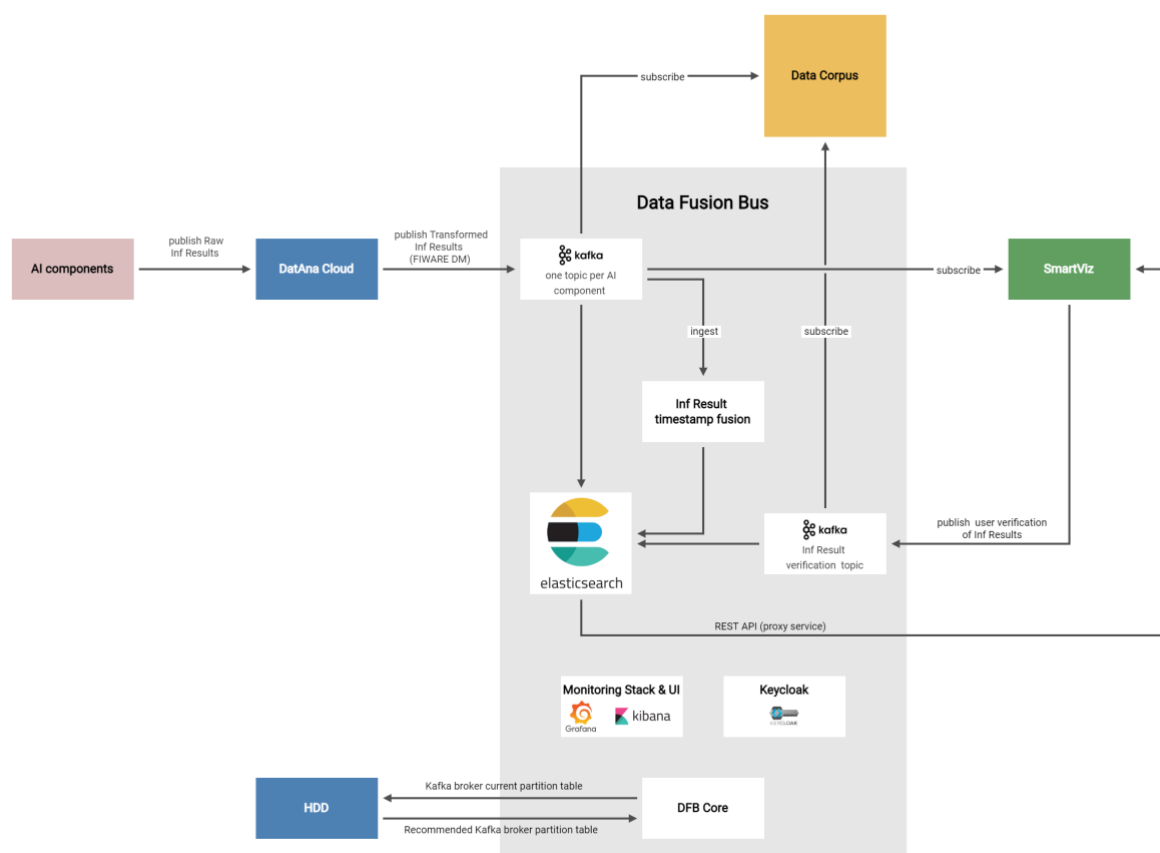


Figure 5: DFB internal architecture and interactions for MARVEL initial version of integrated framework

2.2.4 Connection to other components

According to the system design that was established for the use cases to be implemented for the 1st Integrated Release of the MARVEL framework, the DFB was foreseen to interact with the following components:

- **DatAna.** DatAna receives raw inference results from all AI components and transforms them into FIWARE-compliant counterparts. Subsequently, DatAna publishes the transformed inference results as messages to the DFB Kafka topics that have been configured for AI inference results (one topic per each AI component).
- **SmartViz.** (a) SmartViz receives incoming inference results (FIWARE-compliant) in real-time by subscribing to corresponding DFB Kafka topics. (b) SmartViz publishes inference result verification messages to a dedicated DFB Kafka topic ("InferenceVerification"). The DFB receives these messages in real-time and updates the respective inference results accordingly in the Elasticsearch. (c) SmartViz accesses the DFB REST API to retrieve historical data (FIWARE-compliant inference results) stored in the DFB Elasticsearch.
- **Data Corpus.** (a) Data Corpus receives incoming inference results (FIWARE-compliant) in real-time by subscribing to corresponding DFB Kafka topics. (b) Data Corpus receives inference result verification messages in real-time by subscribing to a dedicated DFB Kafka topic ("InferenceVerification").
- **HDD.** (a) HDD receives from the DFB the currently implemented Kafka topic partition table as a JSON object and a log file with performance metrics. (b) DFB receives from HDD an updated Kafka topic partition table as a JSON object.

2.2.5 Demonstration

Figure 6 illustrates indicative screenshots of the DFB monitoring UI dashboard.



Figure 6: Indicative screenshot of the DFB monitoring UI dashboard

2.2.6 Performance

In order to assess the contribution of DFB to the MARVEL framework, we need to address the following high-level performance indicators: a) Data Integrity, b) Scalability, c) Availability,

and d) Performance for high volume, heterogeneous data streams. Below, a list of specific, measurable metrics is associated with the above indicators, along with a brief description of the purpose of each indicator:

- Data Integrity
 - **Metrics:** Data loss rate.
 - **Description:** Confirm that advanced encryption mechanisms over end-to-end data transfer will guarantee data integrity.
- Scalability
 - **Metrics:** HW speed up.
 - **Description:** Increase the number of modality data streams and verify that performance metrics improve or at least stay the same.
- Availability
 - **Metrics:** Service availability-failed request, data access restriction.
 - **Description:** Verify that DFB resources are available and discoverable.
- Performance for high volume, heterogeneous data streams
 - **Metrics:** Data transfer latency, data throughput, response time, number of cluster nodes.
 - **Description:** Thoroughly measure different performance metrics under different execution conditions.

In the context of the MVP, the performance of the DFB was measured. Table 3 summarises the collected measurements for the specified metrics.

Table 3: Results of the measurements for DFB component

Metric	Value
Data loss rate	0
HW speed up	-
Service availability-failed request	100% availability
Data access restriction	None
Data transfer latency	5 ms (200 MB/s load)
Data throughput	605 MB/s
Response time	5 ms (200 MB/s load)
Number of cluster nodes	3

2.3 StreamHandler

INTRA's StreamHandler Platform provides the hooks for interconnecting, storing, transforming and processing data as well as training, testing and executing machine learning and artificial intelligence algorithms, resulting in a full Big Data solution.

To provide meaningful insights (services), a generic big data solution consists of four main components; data sources, data storage (databases), data streaming and big data management (processing, analytics, visualisation and business intelligence).

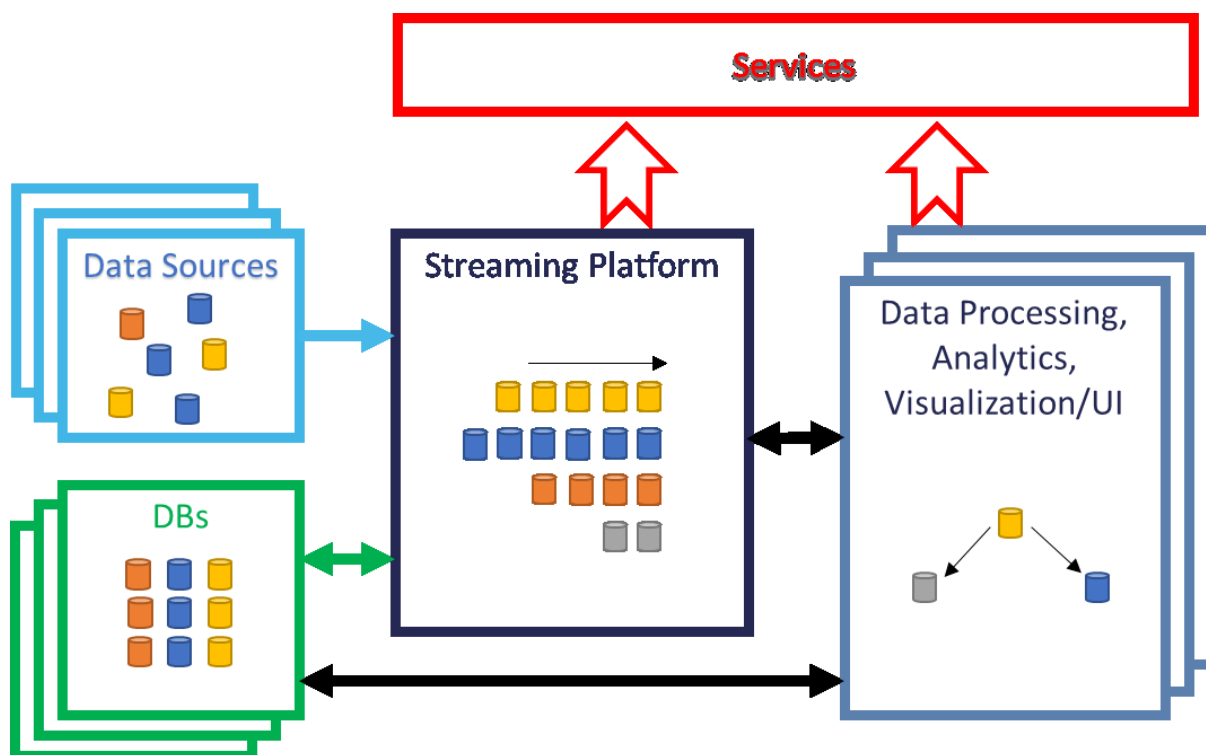


Figure 7: StreamHandler – Conceptual Architecture

INTRA's StreamHandler Platform is a high-performance (low latency and high throughput) distributed streaming platform for handling real-time data based on Apache Kafka. It can efficiently ingest and handle massive amounts of data into processing pipelines, for both real-time and batch processing. The platform and its underlying technologies can support any type of data-intensive ICT services (Artificial Intelligence, Business Intelligence, etc.) from cloud to edge.

In the following paragraphs, an overview of INTRA's StreamHandler Platform is presented, together with a description of the high-level architecture and the individual platform components, their interactions, as well as the key technologies involved.

The key capabilities and features offered by the platform are:

- Real-time monitoring and event-processing
- Interoperability with all modern data storage technologies and popular data sources
- Distributed messaging system
- High fault-tolerance - Resiliency to node failures and support of automatic recovery
- Elasticity - High scalability

- Security (encryption, authentication, authorisation)

In particular, the platform is a fully-featured industrial-grade solution: i) which is capable to scale out and accommodate various and from different domains of big data, interoperating with all modern data storage technologies as well as other persistence approaches and ii) can support all important Big Data languages including Python, Java, R, and Scala as well as other traditional programming approaches.

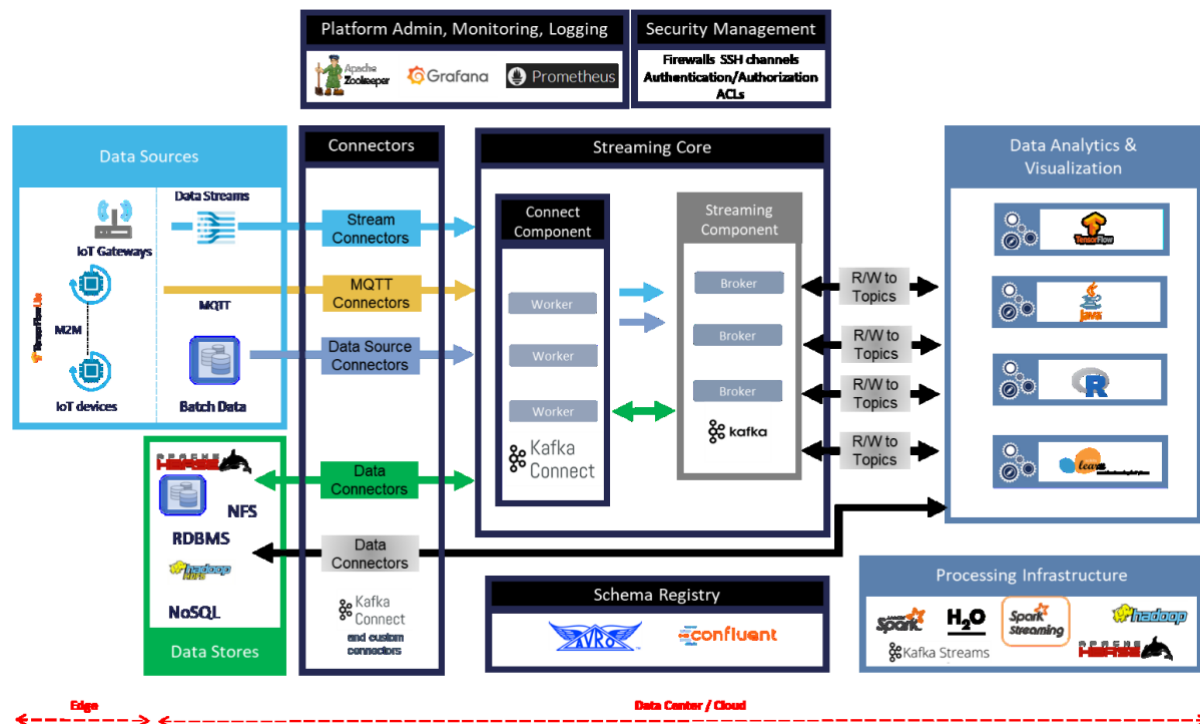


Figure 8: StreamHandler - High level Architecture

As depicted in the figure, INTRA's StreamHandler Platform consists of: i) **Connectors**, ii) **Streaming Core component**, iii) **Schema Registry**, iv) **Security Management**, and v) **Platform Admin and Monitoring Dashboard**. **Data sources, Data stores, Data Analytics and Visualisation applications** as well as the supporting **Processing Infrastructure** are components that complement the offered Big Data solution and their choice and implementation are dependent on the targeted use cases and scenarios. A brief explanation of all components comprising the architecture follows.

a. Data sources and Data stores, which represent **data streams** and **data sources**, both in a structured or unstructured format that can be made available and potentially be connected to the Big Data platform, generated by any IoT device and/or gateway on the edge. Similarly, and according to the requirements, appropriate persistent storage can be used, as depicted in the input/output data components (Figure 8). The described data sources will be seamlessly integrated with processing components by the means of integration connectors (Connectors). The Big Data platform can efficiently interoperate with all the modern data storage technologies of a Big Data ecosystem such as RDBMS, NoSQL, HDFS Hadoop, Apache HBase, etc. as well as other persistence approaches such as Mongo, MySQL, JDBC, etc.

INTRA's StreamHandler Platform can interoperate with IoT architectures building an end-to-end IoT integration with the platform with the use of the MQTT protocol. MQTT is a widely used ISO standard (ISO/IEC PRF 20922) publish-subscribe-based messaging protocol. MQTT has many implementations such as Mosquitto or HiveMQ. MQTT and Apache Kafka are a

perfect combination for end-to-end IoT integration from edge to data centre (and back, of course, i.e., bi-directional).

b. Connectors connect external data sources and make them available to the AI platform. External data sources are connected and made available by employing the “Stream Connectors” and “Data Source Connectors”. Data connectors allow the interconnection of data stores for permanent storage of data. Well-known data stores can be integrated into the platform such as Hadoop, Oracle, JDBC, Neo4j, InfluxDB, Cassandra, MongoDB, Elasticsearch and more. The Confluent Hub offers a huge variety of data connectors for both data stores and sources. In addition, the Kafka Connect API simplifies the integration of a new data source or sink, enabling the development of custom-made connectors.

c. Streaming Core. The Streaming Core component is implemented as an Apache Kafka cluster with multiple brokers to maintain load balancing and replication. The underlying technologies and capabilities of the Streaming Component and the multiple workers of the Connect Component allow the realisation of scalable and secure stream data pipelines. Apache Kafka allows producers and consumers to publish and subscribe to streams of records (topics) similar to the functionality provided by a message queue. Producers push messages into a Kafka topic, while consumers pull messages off of a Kafka topic. The streams of records are stored in a fault-tolerant durable way and consumers (Big Data Apps or AI Apps) can process them as they occur. In general, Kafka is suitable for building real-time streaming data pipelines to reliably get data between systems or applications and for building real-time streaming applications that transform or react to the streams of data. The flexibility of running the Communication Platform in a clustered manner allows for the horizontal scalability of the system achieving thus a scalable, fault-tolerant communication-efficient framework for cross-streaming data management and integration. Kafka Streams partitions the data within a topic using logical entities such as partitions and tasks to achieve data parallelism and enable elasticity, scalability, high performance, and fault tolerance.

d. Schema Registry, which allows the definition and storage of data models describing the data. The Schema Registry is implemented through a Kafka add-on (Confluent Schema Registry) that exposes a RESTful interface for storing and retrieving schemas defined in Apache Avro serialisation format. It stores a versioned history of all schemas, provides multiple compatibility settings and allows the evolution of schemas according to the configured compatibility settings and expanded Avro support. It provides serialisers that plug into Kafka clients that handle schema storage and retrieval for Kafka messages that are sent in the Avro format.

e. Data analytics and Data Visualisation represent the applications that perform the data processing and analytics. These are dependent on the exact use cases that are implemented through the use of the INTRA’s StreamHandler Platform and can be implemented in any programming language typically preferred for data science (such as Python, Java, R, and Scala) or any native programming language (e.g., C/C++, Haskel, etc.).

f. Processing Infrastructure. The underlying infrastructure spans multiple VMs and provides all the necessary technologies and components that enable the storage and analysis of the data involved and further allow the usage of any technology-agnostic algorithms, by providing a distributed computing environment that enables the above. Apache Spark, Hadoop, Kafka Streams, Spark Streaming, and H2O are included among others.

g. Security Management. This includes a set of mechanisms that enhance the security of the platform. Mainly, three components are included, encryption of data through SSL/TLS, authentication through SSL/SASL and authorisation using Access Control Lists (ACLs).

h. Platform Admin Dashboard and Monitoring Tools. These include platform administration and monitoring tools that allow cluster configuration and provide an overview of the cluster performance and health status. The specific tools deployed are Prometheus Monitoring platform, Grafana visualisation, and Zookeeper service. Metrics important to the health status of the platform are scrapped through Prometheus and visualised through Grafana dashboards. The ZooKeeper service is used for managing and coordinating the deployed Kafka brokers. The platform components are capable of exporting JMX metrics which in turn are made available in customised Grafana dashboards. The available information includes the status of the Zookeeper chorus, the status of the Kafka Brokers together with insightful graphs about incoming and outgoing data throughputs, message rates per broker and per topic and other information regarding CPU and memory usage per broker. Similarly, healthy Schema Registry and Rest Proxy instances and their corresponding connections are monitored. The cluster overview is completed by Kafka Connect specific panels which present the user with running Connectors and tasks information in addition to data rates per worker node. The available dashboards allow not only a very good overview of the cluster performance and health status but also provide significant information when performance tuning is necessary.

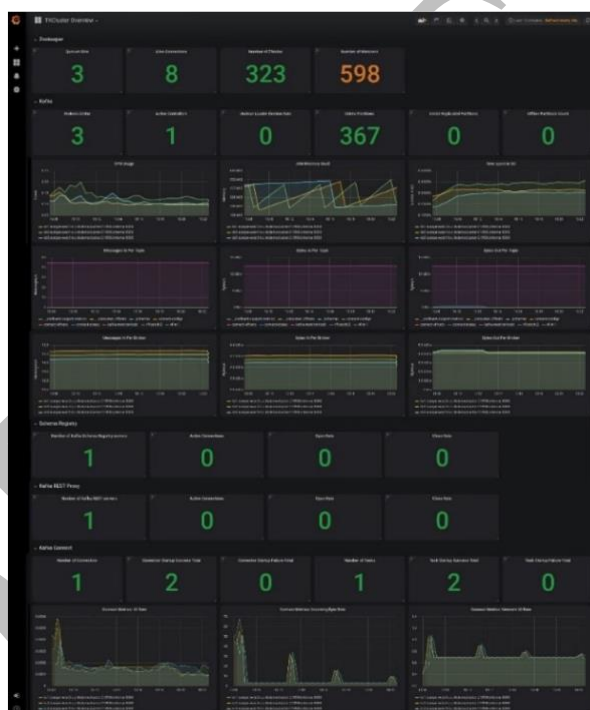


Figure 9: Big Data Platform – Monitoring Dashboard

2.3.1 Progress beyond the state-of-the-art

In the context of the MARVEL framework design activities (WP1) and specification of the DMP (T2.2), certain similarities and overlaps were identified between the functionalities of StreamHandler and those of DFB and DatAna with regards to big data management. However, following an in-depth analysis of the MARVEL framework requirements that the DMP should satisfy, a gap was identified that could not be covered by the DFB and DatAna solutions. This gap was related to the management of audio-visual data. More specifically, the following requirements were established:

- Receive and efficiently archive live streams of audio-visual binary data from all relevant MARVEL sensors, devices and components during system operation.

- The persistent storage of archived AV data should comply with high data security standards and data privacy requirements.
- Provide access to archived audio-visual binary data to the MARVEL UI (SmartViz) by streaming requested archived audio-visual data upon demand in order to present them to the end-user and in association with relevant inference results produced by MARVEL AI components.
- Support the expansion of the data set of the Data Corpus by relaying selected archived audio-visual data to it.

StreamHandler was found to be in a position to be able to satisfy these requirements and fill the gap by extending its supported data source types, connectors and data storage capabilities. This course of action was aligned with INTRA's strategic plan to expand the StreamHandler platform in the direction of audio-visual data management for increased interoperability in order to address additional business cases and reinforce its position in the big data management and smart cities domains.

2.3.2 Design methodology

The requirements that were established for audio-visual data management were analysed and subsequently led to the design and specification of the StreamHandler features.

StreamHandler was designed to be able to connect to AV sources producing raw AV data (e.g., CCTV cameras, network-enabled microphones) and to MARVEL components/services that transmit anonymised audio-visual data (e.g., AudioAnony, VideoAnony). In line with the overall strategy for AV data handling, the Real-Time Transmission Protocol (RTSP) was adopted to connect to AV sources that stream AV data.

In order to achieve increased data security and data privacy levels, the persistent storage of the archived audio-visual data needed to be hosted on an infrastructure that is controlled by the end-user that owns/produces the data. Therefore, StreamHandler was designed to be deployed at the fog layer, where the infrastructure is operated by the end-user. This also aligns with the overall MARVEL E2F2C strategy of maintaining data in close proximity to the source and transmitting only necessary information to higher-level layers.

In order to meet the AV data archiving requirements, an efficient and effective schema was conceived, involving the segmentation of incoming AV data streams into AV files of configurable size based on the specification of the duration of each AV segment. The resulting AV segments were to be stored as binary files in a versatile data store, following a naming convention that can (a) implicitly associate each AV segment with the id of the AV source that produced the corresponding stream and (b) make reference to the segment's absolute start and end times.

In order to meet the requirement of fulfilling on demand requests for AV data from SmartViz, StreamHandler was designed to implement a service that provides a compiled excerpt of a chosen stream provided specific time parameters. The AV data requests would need to include as input arguments: (a) the id of the original AV source that produced the requested AV data, and (b) the absolute start and end times of the requested AV data. StreamHandler was designed to be able to process these requests by (i) retrieving the necessary archived segment(s) that match the input time criteria and (ii) providing the retrieved segments as a single file according to the input time criteria to be consumed as a seamless composite AV stream at the client side.

With regards to relaying collected AV data to the Data Corpus, StreamHandler was designed to be configured to transmit a selection of AV data as binary files to the Data Corpus on a scheduled basis.

2.3.3 Implementation

The implementation of the modules that provide extended functionalities for StreamHandler in order to support the MARVEL use cases for AV data management is based on the following technologies:

- Python 3¹⁸
- rtsp-simple-server¹⁹
- minIO²⁰
- FastAPI²¹
- Docker²²

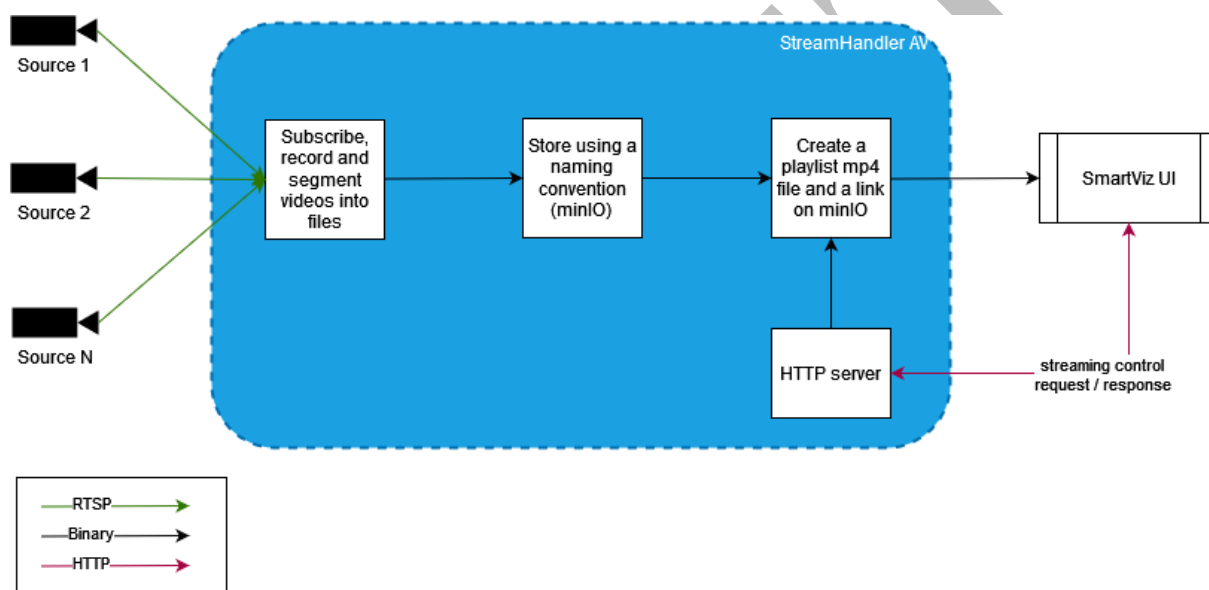


Figure 10: StreamHandler AV implementation

As depicted in Figure 10, StreamHandler AV consists of four distinct services, performing the following tasks:

1. Capturing audio-visual streams arriving from different RTSP sources and segmentation of the files according to a predefined time interval. The ids of the sources, as well as the segmentation parameters, are set in a corresponding JSON file, while the actual RTSP URLs are fetched from the AVRegistry upon initialisation.
2. Storing the audio/video file segments and making them available to external sources via corresponding endpoints. The segmented files are stored in MinIO buckets and are made available to other components for consumption.

¹⁸ <https://www.python.org/>

¹⁹ <https://github.com/aler9/rtsp-simple-server>

²⁰ <https://min.io/>

²¹ <https://fastapi.tiangolo.com/>

²² <https://www.docker.com/>

3. Accepting HTTP requests via a REST endpoint. This is intended to be used by SmartViz and accepts the following parameters: (i) sourceID (ii) timestamFrom and (iii) timestampTo, returning a link to the compiled file that corresponds to the specified source and timeframe.
4. Compiling a unified file upon request and making it available to SmartViz for reproduction. This process is triggered by the aforementioned HTTP endpoint.

2.3.4 Connection to other components

According to the system design that was established for the use cases to be implemented for the 1st Integrated Release of the MARVEL framework, StreamHandler was foreseen to interact with the following components:

- **AVRegistry.** Upon system initialisation, StreamHandler accesses the REST API of AVRegistry to receive information of the AV sources that are active in the current use case in order to connect to them via the RTSP.
- **AudioAnony.** AudioAnony is considered to be an AV source as it generates anonymised AV data. StreamHandler connects to each active AudioAnony instance in the MARVEL network by accessing the RTSP stream URL of the respective instance (this RTSP stream URL is provided as metadata of the available AV sources by AVRegistry).
- **VideoAnony.** VideoAnony is considered to be an AV source as it generates anonymised AV data. StreamHandler connects to each active VideoAnony instance in the MARVEL network by accessing the RTSP stream URL of the respective instance (this RTSP stream URL is provided as metadata of the available AV sources by AVRegistry).
- **SmartViz.** StreamHandler exposes a REST API that accepts incoming requests from SmartViz to transmit on-demand AV data via RTSP. The requests need to include as parameters (a) the id of the original AV source that produced the requested AV data, and (b) the absolute start and end times of the requested AV data.
- **Data Corpus.** StreamHandler transfers binary files containing AV data that have been generated by segmenting the incoming AV data RTSP streams from AudioAnony and VideoAnony.

2.3.5 Performance

StreamHandler was not included in the MARVEL Minimum Viable Product (MVP) version and therefore no performance data are available from that version. Nevertheless, preliminary testing has indicated that StreamHandler is capable of processing at least 3 Full HD AV data streams in parallel with no performance lag when deployed on an infrastructure with 2 CPU cores allocated. Further and more elaborate testing will take place under the specifications and processes set by Task T5.4.

2.4 HDD

HDD is a set of distributed algorithmic schemes for guaranteeing latency requirements [7] while effectively prolonging network lifetime in networked settings [8]. The current MARVEL design of HDD considers the problem of Apache Kafka data topic partitioning optimisation [3]. Apache Kafka uses partitions to scale a topic across many brokers for producers to write data in parallel, and also to facilitate parallel reading of consumers. Even though Apache Kafka provides some out-of-the-box optimisations, it does not strictly define how each topic shall be efficiently distributed into partitions. The well-formulated fine-tuning that is needed in order to improve an Apache Kafka cluster performance is still an open research problem. HDD first models the Apache Kafka topic partitioning process for a given topic. Then, given the set of brokers, constraints and application requirements on throughput, OS load, replication latency and unavailability, HDD formulates the optimisation problem of finding how many partitions are needed and shows that it is computationally intractable, being an integer program. Furthermore, HDD implements two simple, yet efficient heuristics to solve the problem: the first tries to minimise and the second to maximise the number of brokers used in the cluster. As we will demonstrate later on, HDD exhibits the potential to respect the hard constraints on replication latency and perform efficiently w.r.t. to unavailability time and OS load, using the system resources in a prudent way.

A toy example of Apache Kafka topic partitioning is shown in Figure 11. In this example, there is one topic τ , two producers p , four consumers c and four brokers b in the Kafka cluster. The topic has been configured to be partitioned in four leader partitions P with a replication factor of $r = 3$. The objective of HDD is to figure out how to efficiently partition the given topic so as to address the application requirements, but at the same time respect the system constraints.

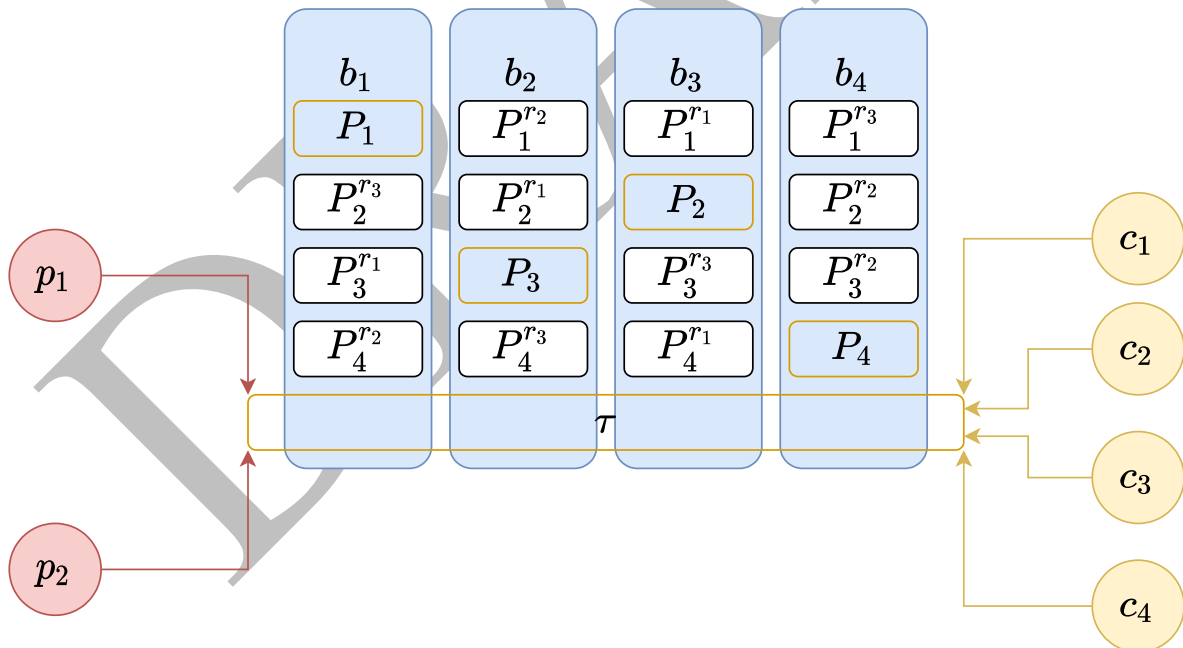


Figure 11: Toy example: Apache Kafka topic partitioning

2.4.1 Progress beyond the state-of-the-art

To the best of our knowledge, there has been no identified study in the state-of-the-art that (a) models the Apache Kafka topic partitioning function in an exact, rigorous manner, (b) identifies, formulates and characterises the underlying combinatorial problem(s), and, (c) designs efficient methodologies via considering the application constraints. Therefore, a

handful of questions come up; for instance, how should a given topic be allocated to partitions? What is an efficient number of brokers to be used? Or, how to satisfy the application requirements? Even if a rigorous approach for topic partitioning in Apache Kafka deployments has not appeared in the related literature, there has been a notable number of works regarding other issues of Apache Kafka operation. We briefly describe those approaches below, in order to highlight the current state of research.

Formal methods have been used to model the communication between producers and consumers in Apache Kafka [9]. In this context, model checking tools verify five properties of the system. The results of verification show the model of data transmission in the Kafka messaging system caters for its specification, from which this system can be concluded that it is reliable. Custom-made Apache Kafka consumers which enhance Apache Kafka with robust, scalable and smart filtering and queuing mechanisms to achieve effective rate adjustment have been also developed [10]. RACER manages to keep the number of re-transmissions to low levels even in high congestion situations while guaranteeing zero dropped data. The experimental results illustrate that RACER is superior to the standard Kafka consumer in terms of stability, consumption rate and throughput. The improvement of fault tolerance in the Apache Kafka pipeline architecture is also promoted by defining optimal checkpoint interval values which impact the data recovery of the original Apache Kafka pipeline [11]. The design is proved to reduce the number of lost data by using the optimal checkpoint interval time. Conforming to the comparative analysis, the modified system improves the performance of data recovery in Apache Kafka. The total overhead cost of Kafka processing decreases by 5% than the original system by applying the fixed checkpoint interval method. Queuing-based packet flow models to predict performance metrics of Apache Kafka cloud services have been proposed [12]. The input arguments of this approach include the number of brokers in the cluster, the number of partitions in a topic and the batch size of data. Using an approach like this, one can improve metrics such as the relative payload and overhead, producer throughput, and the disk storage usage variation over time. Queuing theory is used to evaluate the end-to-end latency of the data.

Decentralised cluster computing on Apache Kafka streams, to improve metrics on seismic waveform data has been effectively delivered [13]. An in-memory distributed complex event recognition engine has already been designed on top of Apache Kafka data flows [14]. Simulation platforms for evaluation of futuristic mobility use cases, built on top of Apache Kafka deployments have appeared in the literature as well [15]. Several approaches break the streaming pipeline into two distinct phases and evaluate percentile latencies for two different networks, namely 40GbE and InfiniBand EDR (100Gbps), to determine if a typical streaming application is network intensive enough to benefit from a faster interconnect [16].

2.4.2 Design methodology

In general, the more partitions there are in an Apache Kafka cluster, the higher the throughput one can achieve. Therefore, a viable objective would be to maximise the number of partitions within the cluster:

$$\max P$$

On both the producer and the broker side, writing to different partitions can be done fully in parallel. In this sense, expensive operations such as compression can utilise more hardware resources. On the consumer side (within a consumer group), Apache Kafka always gives a single partition's data to one consumer thread. Consumers can specify both the topic and partition from which they want to receive data. If consumers don't specify the partition, Apache Kafka will decide and can choose more than one, depending on the number of partitions

available. A topic, however, cannot have more consumers than partitions. Therefore, the degree of parallelism in the consumer is bounded by the number of partitions being consumed:

$$P \geq c$$

We denote the target cluster throughput as T . If the throughput that can be achieved on a single partition for production and consumption (T_p and T_c respectively) is known (through measurements), then we need to have at least the following number of partitions:

$$P \geq \max\left(\frac{T}{T_p}, \frac{T}{T_c}\right)$$

In addition to throughput, there are a few other factors that are worth considering when choosing the number of partitions. In some cases, having too many partitions may also have a negative impact.

Each partition maps to a directory in the file system in the broker. Within that log directory, there will be two files (one for the index and another for the actual data) per log segment. Currently, in Apache Kafka, each broker opens a file handle of both the index and the data file of every log segment. So, the more partitions, the higher the open file handle limit in the underlying operating system must be. Therefore, if H_{max} denotes the maximum number of open file handles that a broker can tolerate, and b the number of brokers, then the number of partitions in the Apache Kafka cluster is bounded by

$$P \cdot r \leq b \cdot H_{max}$$

The end-to-end latency in Apache Kafka is defined by the time from when a message is published by the producer to when the message is read by the consumer. Apache Kafka only exposes a message to a consumer after it has been committed, i.e., when the message is replicated to all the in-sync replicas. So, the time to commit a message can be a significant portion of the end-to-end latency. By default, an Apache Kafka broker only uses a single thread to replicate data from another broker, for all partitions that share replicas between the two brokers. Assuming a given replication factor r , we denote the replication latency as l_r .

Larger Apache Kafka clusters alleviate replication latency. For example, suppose that there are x partition leaders on a broker and that the replication process necessitates in this case y ms of time. If we insert $x/100$ additional brokers in the same Apache Kafka cluster, each of them only needs to fetch $x/10$ partitions from the first broker on average. Therefore, the added latency due to committing a message will be in the order of just $y/10$ ms, instead of y ms. Depending on the application area, the operator might enforce a requirement of a specific (end-to-end, and therefore) replication latency threshold, which we denote as L . Therefore, this threshold bounds the related system configurations as follows:

$$\frac{P \cdot r}{b} \cdot l_r \leq L$$

The intra-cluster replication that Apache Kafka supports, provides higher availability and durability. When a broker fails, partitions with a leader on that broker become temporarily unavailable. Apache Kafka will automatically move the leader of those unavailable partitions to some other replicas to continue serving the client requests. This process is done by one of the Apache Kafka brokers designated as the controller, usually in a serial manner.

In specific cases, the observed unavailability could be proportional to the number of partitions. Suppose that a broker has a total of x partitions, each with y replicas. Roughly, this broker will be the leader for about x/y partitions. When this broker fails uncleanly, all those x/y partitions become unavailable at exactly the same time. Suppose that it takes z ms to elect a new leader

for a single partition. It will take up to $z * x/y$ ms to elect the new leader for all x/y partitions. So, for some partitions, their observed unavailability can be $z * x/y$ ms plus the time taken to detect the failure.

Therefore, given an observed unavailability time u during a broker failure, the application unavailability requirement threshold U shall bound the configuration as follows:

$$\frac{P}{b} \cdot u \leq U$$

Even though Apache Kafka provides some out-of-the-box optimisations, the well-formulated fine-tuning that is needed in order to improve cluster performance is still an open research problem. Based on the equations above, we conclude that the problem is formulated as an integer program. Consequently, the problem is computationally intractable (all of the variables are restricted to be nonzero positive integers, through constraints). This means that we are not able to optimally maximise the exact number of partitions needed to satisfy all the system constraints for any given instance of the problem in polynomial time.

2.4.3 Implementation

The naive way to solve the problem is to simply remove the constraint that P and b are integers, solve the corresponding linear relaxation of the integer program, and then round the entries of the solution to the linear relaxation. But, not only may this solution not be optimal, it may not even be feasible, as it could potentially violate some constraints.

In order to address this challenge, we designed two Algorithms, BroMin and BroMax for a topic partitioning by ensuring adequate throughput, low OS load, low replication latency and high availability. Each of the two algorithms targets at fully exploiting a given broker's resources and at minimising (Algorithm BroMin) or maximising (Algorithm BroMax) the number of brokers used. Although the two approaches seem to follow an opposite line of thought, they both try to maximise the number of partitions needed in the Apache Kafka cluster to satisfy the constraints, but with a different side-objective: use as less HW resources in the first case and take advantage of all the available HW resources in the second case.

More specifically, both algorithms receive a set of parametrical inputs ($T, c, r, H_{max}, L, U, B$) and a set of measured inputs (T_p, T_c, l_r, u). Each algorithm starts gradually increasing or decreasing correspondingly the number of brokers to be investigated, with a lowest number equal to the replication factor r and highest equal to the number of available brokers B . Then, for the selected broker number, both algorithms start decreasing the number of investigated partitions P , starting from the maximum allowed number per broker w.r.t. open file handles, and finishing at the minimum allowed number w.r.t. the desired cluster throughput and consumer support. The algorithms then return as a solution, the first occurrence of P, b combination that satisfies the replication latency and unavailability time constraints. The code of the two algorithms is displayed in Figure 12 and Figure 13.

```

1 function [P, b] = bromin (T, c, r, H, L, U, B, T_p, T_c, l_r, u)
2   min_P = max( [(T/T_p), (T/T_c), c] );
3   for b = r:1:B
4     for P = floor((B*H/r)):-1:min_P
5       if( (P*r*l_r <= b*L) && (P*u <= b*U) )
6         return;
7       endif
8     endfor
9   endfor
10  printf ("No feasible solutions found.\n");
11 endfunction

```

Figure 12: HDD's BroMin algorithm code

```

1 function [P, b] = bromax (T, c, r, H, L, U, B, T_p, T_c, l_r, u)
2   min_P = max( [(T/T_p), (T/T_c), c] );
3   for b = B:-1:r
4     for P = floor((B*H/r)):-1:min_P
5       if( (P*r*l_r <= b*L) && (P*u <= b*U) )
6         return;
7       endif
8     endfor
9   endfor
10  printf ("No feasible solutions found.\n");
11 endfunction

```

Figure 13: HDD's BroMax algorithm code

2.4.4 Connection to other components

According to the system design that was established for the use cases to be implemented for the 1st Integrated Release of the MARVEL framework, the HDD was foreseen to interact with the DFB: (a) HDD receives from the DFB the currently implemented Kafka topic partition table as a JSON object and a log file with performance metrics. (b) DFB receives from HDD an updated Kafka topic partition table as a JSON object.

2.4.5 Demonstration

An example of a JSON object implementation mentioned in the previous subsection is the following:

```

[
  {
    "name": "topic1",
    "internal": false,
    "partitions": {
      {
        "partition": 0,
        "leader": {
          "id": 2,
          "idString": "2",
          "host": "broker1.provider.com",
          "port": 9094,
          "rack": null
        },
        "replicas": {
          {
            "id": 2,
            "idString": "2",

```

```
        "host": "broker1.provider.com",
        "port": 9094,
        "rack": null
      }
    },
    "isr": {
      {
        "id": 2,
        "idString": "2",
        "host": "broker1.provider.com",
        "port": 9094,
        "rack": null
      }
    }
  }
},
{
  "name": "topic2",
  "internal": false,
  "partitions": {
    {
      "partition": 0,
      "leader": {
        "id": 1,
        "idString": "1",
        "host": "broker2.provider.com",
        "port": 9094,
        "rack": null
      },
      "replicas": {
        {
          "id": 1,
          "idString": "1",
          "host": "broker2.provider.com",
          "port": 9094,
          "rack": null
        }
      },
      "isr": {
        {
          "id": 1,
          "idString": "1",
          "host": "broker2.provider.com",
          "port": 9094,
          "rack": null
        }
      }
    }
  }
}
]
```

The initialisation of the simulation environments is as follows:

```

1 clear;
2 clc;
3 interactions = 5;
4 iterations = 1000;
5 printf ("Initializing Apache Kafka configuration.\n");
6
7 %Max numbers
8 B = 20; %max number of brokers
9 H = 10000; %max open file handles per broker
10 U = 2000; %ms - max unavailability time
11 L = 200; %ms - max replication latency
12
13 %System parameters
14 T = 100; %MB/s
15 c = 200*ones(1,interactions); %number of consumers
16 r = 5; %replication rate
17
18 %Measured inputs
19 T_p = 10; %MB/s - FIXED
20 T_c = 20; %MB/s - CHOSEN
21 L_r = 1; %ms - FIXED
22 u = 5; %ms - FIXED
23
24
25 %%Multiple consumer simulations
26 c = [200 300 400 500 600];
27 multiconsim

```

And the experimental function “multiconsim”.

```

1 printf ("Running bromin.\n");
2 P_bromin = zeros(1,interactions);
3 b_bromin = zeros(1,interactions);
4 for i=1:1:interactions
5     [P_bromin(1,i), b_bromin(1,i)] = bromin (T, c(1,i), r, H, L, U, B, T_p, T_c, L_r, u);
6 endfor
7
8 printf ("Running bromax.\n");
9 P_bromax = zeros(1,interactions);
10 b_bromax = zeros(1,interactions);
11 for i=1:1:interactions
12     [P_bromax(1,i), b_bromax(1,i)] = bromax (T, c(1,i), r, H, L, U, B, T_p, T_c, L_r, u);
13 endfor
14
15 printf ("Running MS-CNFL.\n");
16 P_mscnfl = zeros(1,interactions);
17 b_mscnfl = B*ones(1,interactions);
18 for i=1:1:interactions
19     P_mscnfl(1,i) = floor( min( mean(randi([1 1000*B/r],1,iterations)), mean(randi([1 100*B],1,
20     b_mscnfl(1,i) = floor( mean(randi([1 B],1,iterations)) );
21 endfor
22
23

```

2.4.6 Performance

For the purposes of evaluating the efficiency of HDD, we took into account the industrial best practices in the related application sectors. We identified Kafka setup guidelines used by

credible industrial service providers. For example, Microsoft, recommends that it would be better to constrain the existing partitions per broker (including replicas) to a number, not more than 1,000. In another example, Confluent recommends setting the number of partitions per broker to at least $100 \cdot B$. Consequently, combining the essence of these configuration recommendations, we arrive at the following benchmark method, called MS-CNFL: $P = \min \left(P \in_R \left[1 \dots \frac{1.000 \cdot B}{r} \right], P \in_R [1 \dots 100 \cdot B] \right)$ and $b \in_R [1 \dots B]$, where \in_R denotes uniformly random selection. We measure the system throughput, captured by the ultimate number of partitions selected by each algorithm, the replication latency, captured the amount of time that is needed to process each message, in the sense of time required for data to be stored or retrieved, the numbers or costs of the application's infrastructure, captured by the number of brokers used in the Apache Kafka cluster, the OS load metric via the open file handles and the unavailability metric via the unavailability time. We perform the measurements for a variable number of consumers.

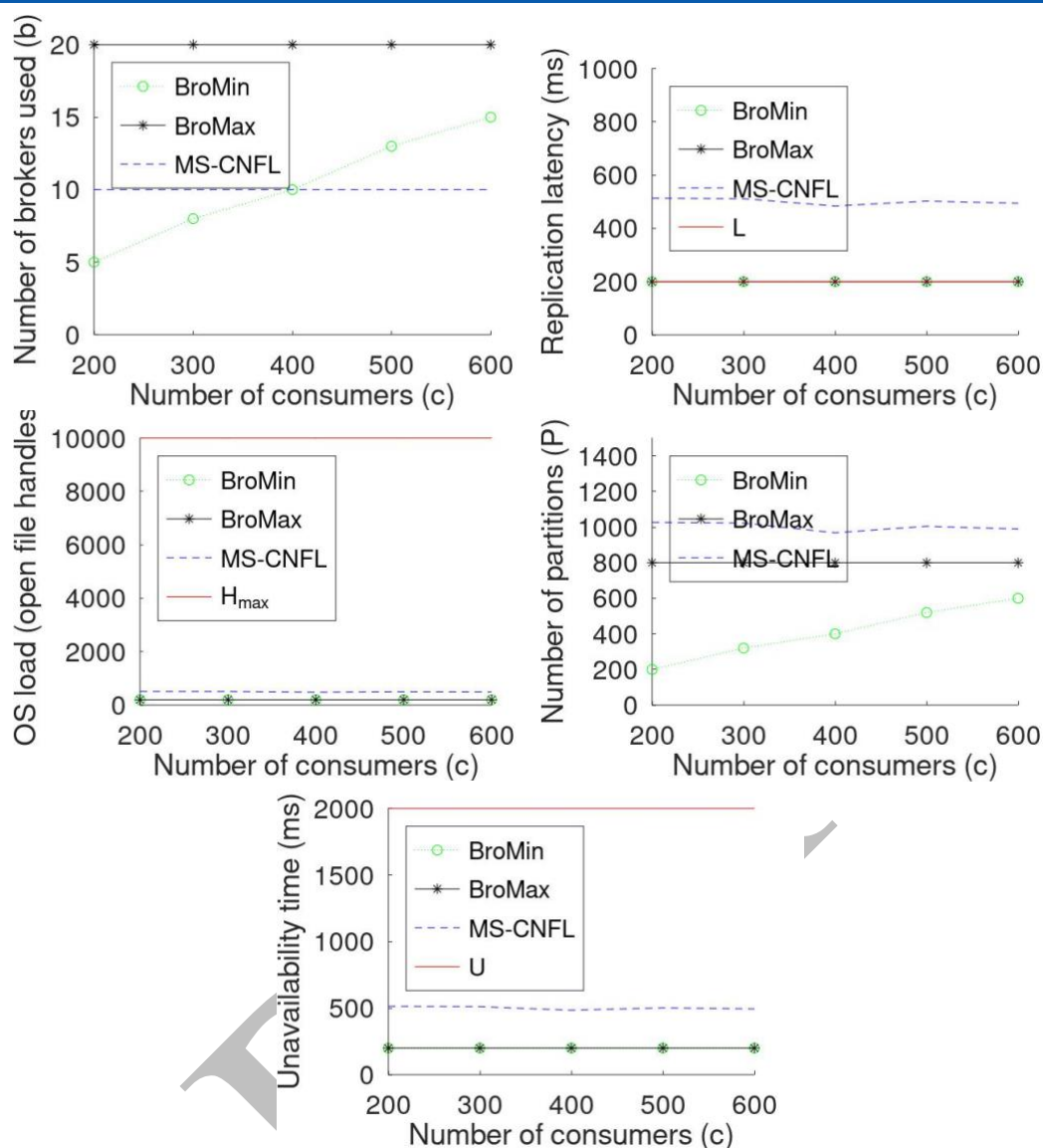


Figure 14: HDD simulation performance

Figure 14 shows the five considered metrics for different consumer number values. BroMax and MS-CNFL keep a steady partitioning number, independently of the consumer number, with MS-CNFL keeping more partitions. On the other hand, BroMin increases the partitions similarly to the increase of the consumer number. An equivalent phenomenon can be seen when we take into account the broker number metric. BroMax and MS-CNFL keep a steady number of brokers, independently of the consumer number, and BroMin increases the brokers similarly to the increase of the consumer number. In this case, we can see that BroMin is choosing to use more brokers than MS-CNFL for more than 400 consumers in the system. When it comes to the application constraints, we observe that although BroMin and BroMax respect them all, MS-CNFL violates the latency one. This behaviour is due to the fact that MS-CNFL keeps a larger partitioning number than BroMin and BroMax, but tries to distribute the partitions to a less efficient broker number/set. This option is even more highlighted in the case of >400 consumers, in which even BroMin utilises more brokers than MS-CNFL. We can also see that in the instances when the hard constraints are not violated by MS-CNFL (OS load and unavailability), BroMin and BroMax lead to a more efficient performance when it comes to partitions.

3 Data Corpus and augmentations

One of the main objectives of the MARVEL project is to create a Big Data repository with datasets from smart city environments and share it with research and industrial communities. Data Corpus materialises this goal. Anonymised and annotated data from the pilots are stored in the Corpus, which constitutes a clustered Big Data repository. Data can be uploaded either on-demand by the pilot users through a web-based interface or automatically by other MARVEL components (i.e., StreamHandler can transmit the main audio/video data files and DFB can publish inference results). The MARVEL user can also apply augmentation techniques to the uploaded datasets and create variations of them that are meaningful for Machine Learning (ML) operations. The *internal MARVEL components* (i.e., DFB-SmartViz) can enhance their operation by further processing all pieces of data. *External users* can also access the datasets that have been authorised for public use by the pilots, via a graphical interface. The external user can view the available datasets and their underlying snippets, as well as perform queries to find the subset of data that he/she requires (e.g., based on keywords, pilot/contributor name, etc.). A high-level view of the overall setting is depicted in Figure 15. For the time being, these are the main operations and interactions that are considered for the Corpus. A preliminary implementation and integration will be provided for the first prototype of MARVEL, while further elaborations will be performed during the second phase of the project.



Figure 15: The main Data Corpus data flow

The unified data model of the Corpus and the augmentation operations are the main subjects of task “T2.3: Incremental scheme: continuous augmentation of the dataset”. The main Corpus activity is part of the task “T2.4: Sharing multimodal Corpus-as-a-Service: fostering the European data economy vision in smart cities”, which starts at M20. One of the objectives of this task that will be elaborated during the second half of the project is to develop a primary SLA with respect to relevant aspects, like accessibility, operability, managing streaming and network, legal considerations, security, privacy, and technical concerns.

3.1 Realisation of the MARVEL Data Corpus

There are several technologies that are utilised for the realisation of the Corpus. These include server-side technologies that materialise the main storage operations, as well as client applications and interfaces that other users/components can utilise in order to interact with it.

Storing of the audio/video files themselves is performed by the Hadoop Distributed File System (HDFS). It saves files on commodity machines, providing high aggregate bandwidth across the Corpus cluster. The administration of this repository is supported by the HBase distributed database (an open-source non-relational distributed database). The management application of this database is implemented by the Zookeeper. It is an open-source module that provides administrative services, like maintaining naming and configuration information and providing distributed synchronisation, etc. Thereupon, queries (accessing) are performed via the web interface of Ambar that offers an intuitive, easy-to-use Hadoop management solution with Web UI and RESTful APIs. The overall integration of these technologies is illustrated in the following figure.

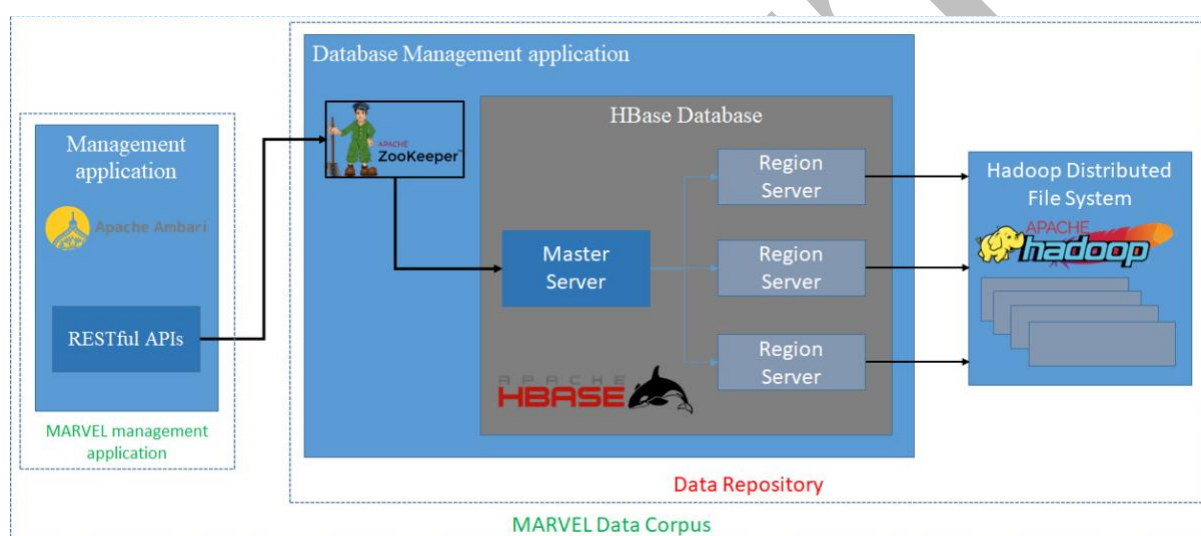


Figure 16: Internal architecture for the core MARVEL Data Corpus repository

Also, JAVA client applications are implemented, which can programmatically call the server-side APIs and interact with the core repository. These include: i) an application that reads files from a folder and ingest them in the Corpus – integration with StreamHandler; and ii) an application that connects dedicated Kafka topics and ingests inference results that are published there by the MARVEL AI components – integration with DFB.

3.2 Data augmentation strategy

Another valuable feature of the Data Corpus involves the incorporation of augmentation techniques. It is ordinary in the ML field to augment the original datasets in an attempt to enhance the categorisation (or other) capabilities of the learnt models. For instance, if one has collected a video dataset from a public square during the morning, he/she can create an augmented version of this dataset by applying brightness filters, simulating the same use cases in the afternoon. Thus, the enhanced ML model is expected to perform better when it will evaluate a real video stream during the afternoon hours, than the original one that has been trained solely with the initial raw data. The Corpus supports several augmentation techniques, both for video and audio files, trying to simulate different timepoints within the day or different weather conditions (e.g., apply a video filter to simulate rain).

In order to achieve the goal of making 3.3PB of data available through the Corpus, a continuous augmentation of the data that are ingested into the Corpus must be performed. Therefore, under T2.3 an augmentation strategy is developed including state-of-the-art techniques in terms of augmenting audio/video data. Based on this strategy, several *Python-based augmentation scripts* were deployed and tested using *Keras*²³, *TensorFlow*²⁴, and *imgaug*²⁵ *open-source software libraries* to automatically augment the datasets that are contributed by the pilots. Following some temporary data-related constraints, a portion of data has been injected and augmented into the Corpus. It is expected that MARVEL data providers will bypass the difficulties and achieve the target of 3.3 PB of data within Y2 and Y3.

As described above, several Python-based augmentation scripts were developed in order to automatically augment the datasets provided by the pilots. For image-based tasks, we used the library of *imgaug*, which supports a large range of augmentation techniques, and allows to easily combine these and execute them in random order or on multiple CPU cores.

An example of a video-based augmentation script is provided in Figure 17. The script takes as arguments the input folder where the original non-augmented files reside, the output folder where the augmented files will be produced and the number of clips that will be generated for each input. The script also takes care of the name production of the augmented files while it prints for each augmentation the respective augmentation parameters.

```
import cv2
import os
import imgaug as ia
import imageio
from imgaug import augmenters as iaa
import shutil
import argparse
import random
import time
from vidaug import augmentors as va
from multiprocessing import Pool
import concurrent.futures

main_folder=""
output_folder=""
no_of_clips_to_augment_per_frame=0
video_clip_names=[]

def augment_and_save_frames(video_reader,output_folder,video_clip_name,i,fps,w,h):
    """
    Fetch each frame of video and augment and save as picture in a temporary folder
    Args:
        video_reader: The Video reader object
        rotation_angle: int (Angle of rotation of image)
        noise_value: int (noise value between 0 to 100)
```

²³ <https://keras.io/>

²⁴ <https://www.tensorflow.org/>

²⁵ <https://imgaug.readthedocs.io/en/latest/>

```
temp_folder_path: string (temporary path to store video frames)
output_folder: string (output folder path)
video_clip_name: string (video name)
i: no of clip augmented
"""

# Removing the potential abnormalities in file name
temp = video_clip_name.replace(" ", "")
temp = temp.split(".")
editted_name = temp[0]+"_"+str(i)+"."+temp[1]
path_of_video_to_save = output_folder+"/"+editted_name

# Setting the Noise value for video augmentation
noise_value = random.randint(0,60)
# noise_value = 0
if i%2==0:
    flip=True
else:
    flip=False

# Setting the Rotation angle for video augmentation
rotation_angle = random.randint(-30,30)

# Printing the augmentation parameteres
print("Rotation angle for augmented clip is :", rotation_angle)
print("Noise value to add to augmented clip is :", noise_value)
print(editted_name, rotation_angle, "degrees")

# Perform the augmentation
seq = iaa.Sequential([
    iaa.Fliplr(flip),
    iaa.Affine(rotate=rotation_angle),
    iaa.AdditiveGaussianNoise(scale=(0, (noise_value/100)*255))
])

# Type of video codec, currently is set to mp4
fourcc = 'mp4v' # output video codec
video_writer = cv2.VideoWriter(path_of_video_to_save, cv2.VideoWriter_fourcc(*fourcc), fps, (w,h))

try:
    while video_reader.isOpened():
        ret, frame = video_reader.read()
        if not ret:
            break
```

```
        image_aug = seq(image=frame)
        video_writer.write(image_aug)

    except Exception as e:
        print(e)

# Close all threads of cv
cv2.destroyAllWindows()
video_reader.release()
video_writer.release()

def augment_videos(i):
    """
    Args:
        i: no of clip augmented
    """
    try:
        video_path = f"{main_folder}/{video_clip_names[clip_no]}"
        print(video_path)
        video_reader = cv2.VideoCapture(video_path)

        fps = int(video_reader.get(cv2.CAP_PROP_FPS))
        w = int(video_reader.get(cv2.CAP_PROP_FRAME_WIDTH))
        h = int(video_reader.get(cv2.CAP_PROP_FRAME_HEIGHT))
        # Get the fps for input video and printed out
        print(f"FPS of {video_clip_names[clip_no]} is {fps}")
        start = time.time()
        augment_and_save_frames(video_reader,output_folder,video_clip_names[clip_no],i,fps,w,h)
        end = time.time()
        print("Total augmentation time for single video :", end-start)
    except Exception as e:
        print(e)

time_of_code = time.time()
if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('--main-folder-path', type=str,default='',help='This is the folder path that contains series of video files to be augmented',required=True)
    parser.add_argument('--output-folder-path', type=str, default='',required=True,help='This is the folder path that will contain the augmented video files')
    parser.add_argument('--max-clips', type=int,required=True,help='The maximum number of clips to be augment per input video file.')

    opt = parser.parse_args()
    print("Args \n",opt)
    main_folder=opt.main_folder_path
    output_folder=opt.output_folder_path
    no_of_clips_to_augment_per_frame = opt.max_clips
```

```

# Printing some usefull information
print("Output folder path", output_folder)
print("Main folder path", main_folder)
print("Max augmented clips", no_of_clips_to_augment_per_frame)

if os.path.exists(output_folder) and os.path.isdir(output_folder):
    shutil.rmtree(output_folder)
os.makedirs(output_folder, exist_ok=True)

video_clip_names = os.listdir(main_folder)
print(f"Video files found on input folder are : {video_clip_names}")
no_of_clips_available = len(video_clip_names)

# Run for each clip that needs to be augmented
for clip_no in range(no_of_clips_available):
    # Rotate the clip based on angle range and increment the subsequent clips w.r.t. the angle increment
    print("Number of videos files to be augmented :", no_of_clips_to_augment_per_frame)
    with concurrent.futures.ThreadPoolExecutor() as executor:
        executor.map(augment_videos, list(range(no_of_clips_to_augment_per_frame)))

end_time = time.time()
print("Total augmentation time :", end_time-time_of_code)

```

Figure 17: An example of a video augmentation script. The latter will rotate the input video file towards a randomly generated angle and add a randomly Gaussian noise to the input file. The final output files will be also flipped

The latter python script can be executed through the bash line engaging Python3, as showed in Figure 18. It takes as arguments: a) the input folder path where the original files reside, b) the output folder path where the augmented file will be produced, and c) the number of augmented files that will be produced per videos input.

```

usage: video_augmentation_code.py [-h] --main-folder-path MAIN_FOLDER_PATH --
output-folder-path OUTPUT_FOLDER_PATH --max-clips MAX_CLIPS

video_augmentation_code.py --main-folder-path videos/test --output-folder videos/out/ --
max-clips 5

```

Figure 18: Usage example of the augmentation script in python with the respectively input parameters

The supported augmenters that have been tested and can be used in the augmentation script are summarised in Table 4.

Table 4: The augmenters that have been used and tested for the augmentation process

a/a	Name of the augmenter	Description
1	Gaussian noise	Add Gaussian noise to video
2	Brightness	Change the brightness of a video
3	Invert	Invert colors of a video

4	Grayscale	Converts to grayscale
5	Sharpen	Sharpen the video
6	LinearContrast	Improve or worsen the contrast of the video

By setting the flipping parameter to false and adding only Gaussian noise to the input files (as described in Figure 19), and suppressing any geometric transformation, we can produce new augmented video files by executing the python script in bash as follows:

```
video_augmentation_code_noise.py --main-folder-path videos/test --output-folder videos/out/ --max-clips 10
```

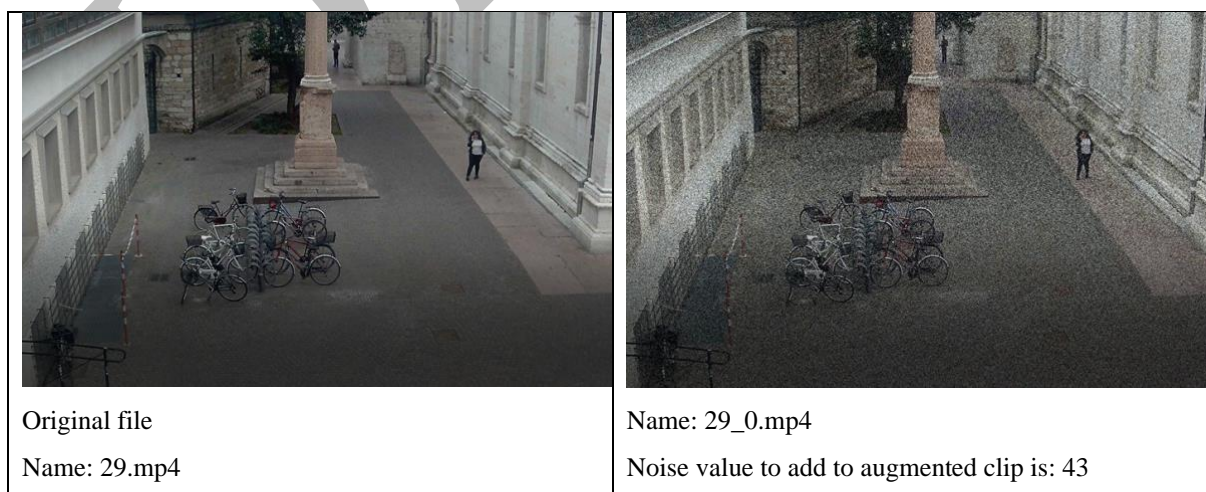
The input file that has been used for this example is from the stage recording of Piazza Maggiore by FBK, placed in the shared folder of MARVAdash (FBK_MT_stage_recording_piazza_maggiore/dangerous_situation/29.mp4).

```
...
# Perform the augmentation
seq = iaa.Sequential([
    iaa.Fliplr(False),
    iaa.AdditiveGaussianNoise(scale=(0, (noise_value/100)*255))
])
...
```

Figure 19: A snapshot of the augmentation script where the augmentation technique that is used is only the Gaussian noise and with no flipping

The augmented videos (as snapshots) of the above executed script are summarised in Table 5.

Table 5: Snapshots of the augmented files produced, based on input file from the stage recording of Piazza Maggiore. The produced augmented files have been set to 10, while the Gaussian noise that has been injected is created randomly





Name: 29_1.mp4

Noise value to add to augmented clip is: 57



Name: 29_2.mp4

Noise value to add to augmented clip is: 40



Name: 29_3.mp4

Noise value to add to augmented clip is: 60



Name: 29_4.mp4

Noise value to add to augmented clip is: 15



Name: 29_5.mp4


Noise value to add to augmented clip is: 54



Name: 29_6.mp4

Noise value to add to augmented clip is: 42

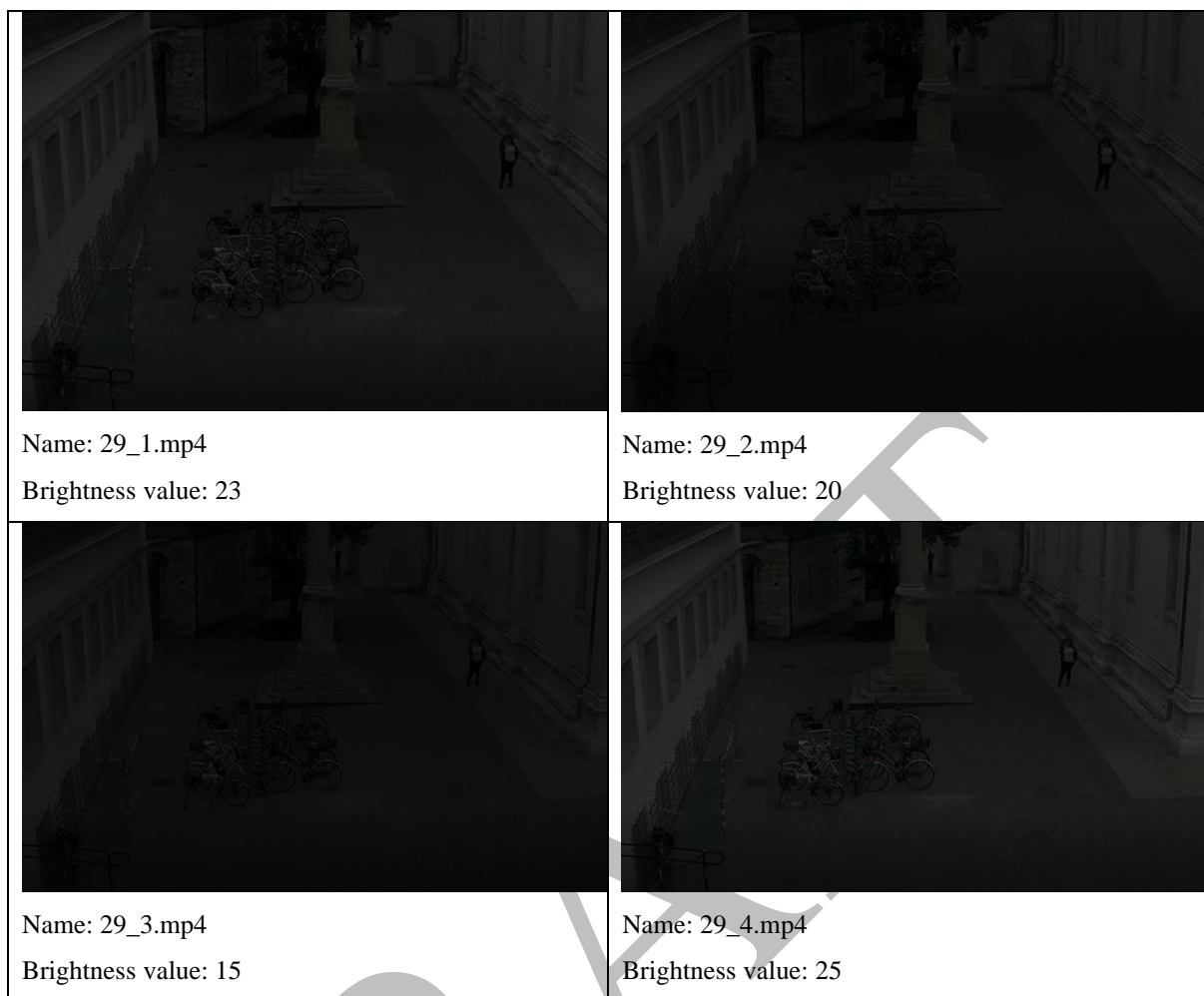


Name: 29_7.mp4 Noise value to add to augmented clip is: 10	Name: 29_8.mp4 Noise value to add to augmented clip is: 46
	
Name: 29_9.mp4 Noise value to add to augmented clip is: 56	

The following results, presented in Table 6, were produced when setting the totally produced augmented clips to the value of 5 and applying the brightness augmentation script with random values of brightness. Once more, the input file that has been used for this example is from the stage recording of Piazza Maggiore, placed in the shared folder of MARVAdash (FBK_MT_stage_recording_piazza_maggiore/dangerous_situation/29.mp4).

Table 6: Snapshots of the augmented files produced, based on input file from the stage recording of Piazza Maggiore. The produced augmented files have been set to 5 while the brightness value applied to each produced file has been created randomly.

	
Original file Name: 29.mp4	Name: 29_0.mp4 Brightness value: 29



3.3 The graphical user interface

The user (either MARVEL internal or external) can access the Corpus via a graphical user interface (GUI). This GUI is developed in Angular 12 and can be utilised by users to review and retrieve the ingested datasets. A user can also perform queries and search for datasets/snippets (e.g., pilot name, based on keywords, etc.). Underneath, such queries are facilitated by the Elasticsearch-Logstash-Kibana (ELK) stack and the user performs them from the GUI. In the context of the MARVEL Data Corpus UI prototyping process, several different user stories have been considered, such as viewing, adding, and deleting the data. However, the ultimate goal to perfectionate a given user journey is one: to have the best UX and via a few clicks to be able to process data of the Corpus.

An indicative user flow starts with the main front page of the UI where the MARVEL Data Corpus user can have an overall overview of the current status of the uploaded datasets and snippets. Furthermore, the latter works as a starting point for the user to add any new dataset, view, or update existing ones. The important feature of this flow is that the user can have, a single page, a complete synopsis of the uploaded data inside the Corpus and the corresponding actions over them.

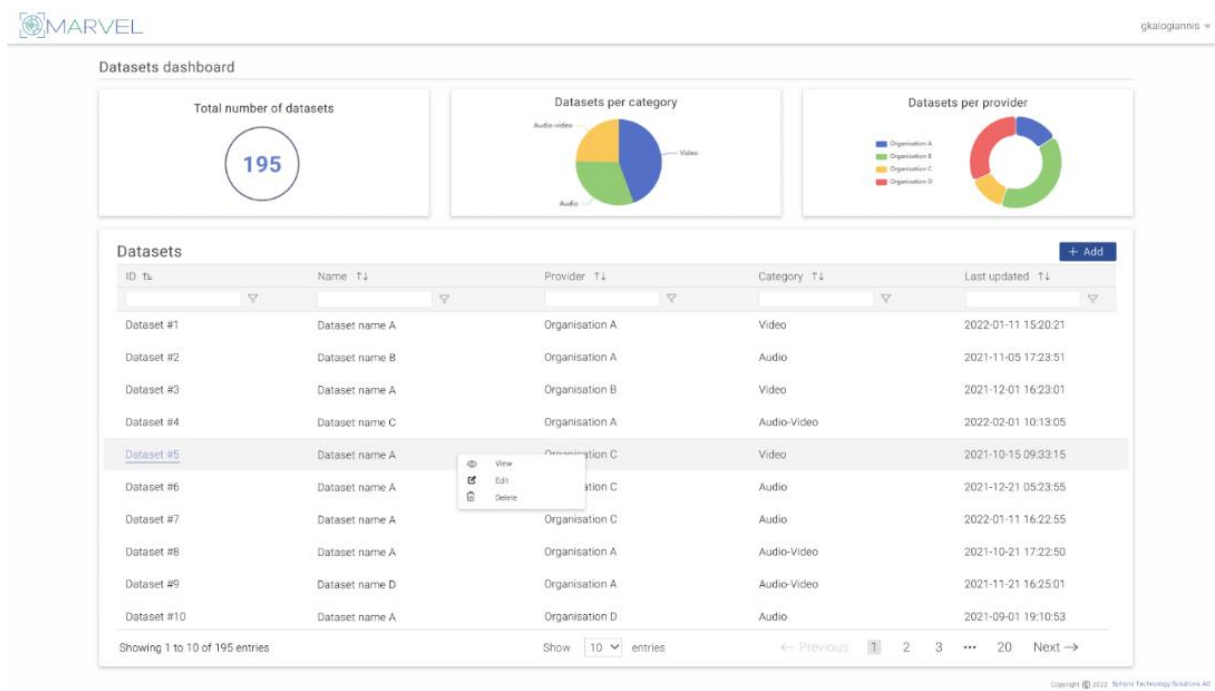


Figure 20: Corpus GUI – Main Dashboard (mock-up data)

From this point, the internal MARVEL user can add, edit, view, or delete the selected dataset with a simple click since the main page of the UI will redirect him/her to the corresponding page of the interface. When it comes to adding data to the Corpus, the interface will guide him/her via a single page where a series of related fields must be filled (in accordance with the unified data model of the dataset entries in the Corpus repository).

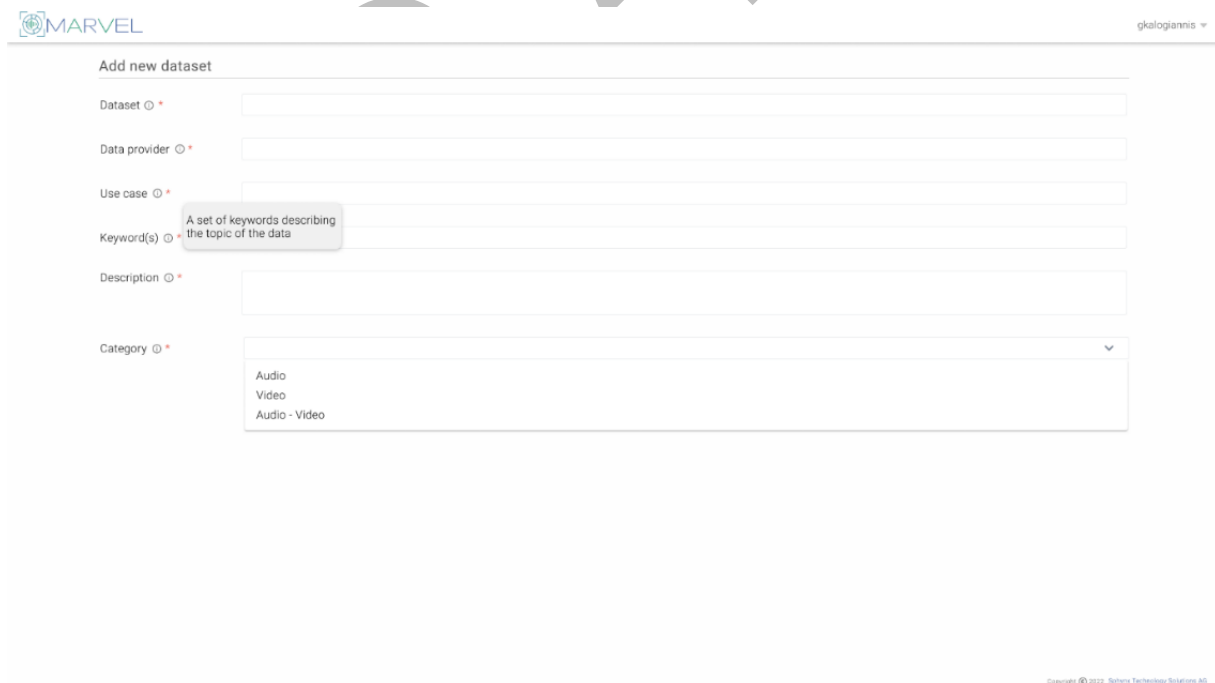


Figure 21: Corpus GUI – Add new dataset

Editing a dataset is as simple as it can be and can be done through a single page also. Since the relative correlated information needed to be filled by the user is quite a lot, the UI of the Corpus, via a uniform view, gives the ability to have an overall control of his/her entries.

gkalogiannis ▾

Edit dataset: ID-53123

Dataset

Data provider

Use case

Keyword(s)

Description

Category

Metadata *

Audio bitrate (Kbps) Audio sampling (kHz)

Number of channels Annotation Software

Annotation ontology Duration (sec)

Location latitude Location longitude

Device id

Snippet(s) *

Snippet #1 🗑️ ⬆️

Publication date Duration (sec)

Start time (sec) End time (sec)

Timestamp Is annotated

Annotator id Annotation file

Annotation summary Additional events

Is augmented Augmentation method

Is anonymized Anonymization method

Snippet file

Doppler © 2022. Software Technology Solutions AG

Figure 22: Corpus GUI – Edit dataset

Last but not least, the MARVEL Data Corpus user can view and delete a specific dataset by just selecting it and performing the relative action. Upon successful deletion, the dataset list presented on the front page of the UI will be automatically refreshed.

The overall Corpus and its GUI functionality are being designed to achieve maximum interaction with all pilots and their end-to-end demonstrations, subject to the cloud data sharing restrictions. There might be access restrictions for some piloting data that cannot be offered for use by the general public outside the consortium members.

4 Positioning within the E2F2C continuum

4.1 The DMP toolkit architectural approach

The DMP toolkit is applied in the AI inference pipeline of MARVEL and is adapted to the needs of each particular use case addressed by MARVEL. The main functionalities can be described as follows.

Summary: DatAna is responsible for collecting the raw inference results from all AI components from all layers through its instances residing at each layer, transforming them into FIWARE-compliant counterparts and then transferring them to higher layers in the E2F2C continuum. The DatAna cloud layer thus aggregates all transformed inference results and relays them to the DFB, which also resides in the cloud. The DFB persistently stores all FIWARE-compliant inference results it receives, but also makes them available in real-time to SmartViz and Data Corpus. The DFB also exposes a REST API to SmartViz to allow it to access all archived inference results in its Elasticsearch database. In addition, the DFB and Data Corpus receive user-generated inference result verification messages from SmartViz and use this information to update the corresponding inference results stored in their databases. HDD interacts exclusively with the DFB to receive information on current Kafka topic partitioning and associated performance metrics and to send updated, optimised Kafka topic partitioning recommendations. In parallel, StreamHandler receives information on active AV sources (CCTV cameras, network-enabled microphones, AudioAnony, and VideoAnony instances) after requesting it from the AVRegistry via a REST call and uses that information to connect to all active AV sources and receive their AV data streams to segment them and store them persistently. StreamHandler also exposes a REST API that is accessed by SmartViz to request archived AV data from specific sources and points in time.

DatAna is a component that is distributed across all three E2F2C tiers, with a separate instance deployed at each infrastructure node. DatAna is complemented by an MQTT message broker, which is also deployed at each infrastructure node, alongside DatAna. Each instance of the MQTT message broker is responsible for collecting raw inference results from the AI components residing on the same tier as the respective MQTT instance. Specifically, AI components publish their raw inference results to dedicated MQTT topics in real-time as they are being produced through the analysis of the AV data streams they receive. The raw inference results of each AI component are formatted as JSON documents according to a dedicated distinct data model that fits the requirements of each AI component. However, all data models of raw AI inference results, include the following information in dedicated fields:

- **AV source id.** The id of the AV source that produced the stream that was analysed to produce the inference result.
- **Inference result id.** A unique identifier for the inference result.
- **Timestamps.** In case the inference result refers to an instant in time a single timestamp is provided. In case the inference result refers to a period in time, two timestamps are provided, corresponding to the start and end of the time period of the result. All time information is absolute and follows the ISO8601 UTC format.

Besides the above, the raw inference results contain other fields that are specific to the needs of each AI component.

Each DatAna instance residing on the same infrastructure node as an MQTT broker subscribes to the broker's topics to receive all incoming raw AI inference results. Subsequently, DatAna transforms the raw inference results into FIWARE-compliant counterparts. Specifically, three

data models that belong in the collection of smart data models of the FIWARE standard have been identified to be relevant to MARVEL. These are:

- **MediaEvent**²⁶. Used to describe general AI inference results.
- **Alert**²⁷. Used to describe AI inference results that should be perceived as alerts.
- **Anomaly**²⁸. Used to describe AI inference results that should be perceived as detected anomalies.

For the needs of MARVEL, the aforementioned data models have been modified by adding additional fields to account for the project's needs, but all existing fields specified by the FIWARE standard have been preserved and are used when relevant, so as to maintain compliance with the standard. Based on the nature of the raw inference result it receives, DatAna selects the most appropriate data model to perform the transformation.

The information that DatAna collects is relayed to higher-level layers of the MARVEL E2F2C continuum. Specifically, the FIWARE-compliant inference results collected by DatAna at the edge layer are relayed to DatAna at the fog layer and the FIWARE-compliant inference results collected by DatAna at the fog layer are relayed to DatAna at the cloud layer. The DatAna instance at the cloud layer is responsible for relaying the FIWARE-compliant inference results it collects from all layers to the DFB by publishing them to the appropriate DFB Kafka topics.

The **DFB** resides in the cloud and receives all FIWARE-compliant inference results published by the DatAna cloud instance and stores them persistently in its Elasticsearch database. The DFB also exposes a REST API to SmartViz to allow it to access all archived inference results in its Elasticsearch database. The DFB receives user-generated verifications of inference results from SmartViz when they are published to a dedicated DFB Kafka topic and uses them to update the respective archived inference result entries accordingly. The DFB also accesses a REST API at the HDD for dispatching the currently applied Kafka topic partition information along with associated performance measurements to it. Using the same REST API, the DFB can also receive updated Kafka topic partition allocation that is recommended by the HDD.

SmartViz is subscribed to all DFB Kafka topics where DatAna publishes FIWARE-compliant inference results to receive them in real-time and present them to the user. SmartViz also allows users to verify the inference results they are presented with. SmartViz transmits these user-generated verifications to the DFB by publishing them to a dedicated Kafka topic available at the DFB.

The **Data Corpus** resides in the cloud and is subscribed to all DFB Kafka topics where DatAna publishes FIWARE-compliant inference results to receive them in real-time and archive them internally to make them available for further AI training purposes along with the associated AV data it collects from StreamHandler. The Data Corpus is also subscribed to the DFB Kafka topic that is used by SmartViz to publish user-generated inference result verifications to receive them in real-time and update the corresponding archived inference results accordingly. The Data Corpus is also connected to StreamHandler, from which it receives AV data as binary files that are a result of AV stream segmentation.

StreamHandler resides in the fog and receives AV data streams from all active AV sources (CCTV cameras, network-enabled microphones, AudioAnony, and VideoAnony instances) via

²⁶ <https://github.com/smart-data-models/dataModel.Multimedia/blob/master/MediaEvent/README.md>

²⁷ <https://github.com/smart-data-models/dataModel.Alert/blob/master/Alert/README.md>

²⁸ <https://github.com/smart-data-models/dataModel.Alert/blob/master/Anomaly/README.md>

RTSP. During initialisation, StreamHandler accesses the REST API of the AVRegistry to discover the active AV sources and their details. During operation, StreamHandler consumes the AV RTSP streams and segments them according to pre-specified time intervals to generate binary documents, suitable for persistent storage. StreamHandler archives the generated AV data files and also exposes a REST API to accept requests from SmartViz about the transmission of AV data from specific AV sources (reference to AV Source id) and from specific points in time. Upon such requests, StreamHandler retrieves the necessary binary files, compiles a unified/edited version of the stream that corresponds to the timeframe requested and generates a link to the said binary file which is to be consumed by SmartViz.

The **HDD** exposes a REST API to allow the reception of the currently applied DFB Kafka topic partition information along with associated performance measurements from the DFB. The HDD uses this information as input to calculate an optimised Kafka topic partition allocation and subsequently makes it available to the DFB via its REST API.

The DMP has been applied in 5 use cases defined for the needs of the initial version of the MARVEL Integrated framework. The following figures illustrate the system architecture diagrams that were specified to address these 5 use cases, which include the specific configuration of the DMP toolkit for each one along with the associated DMP components (depicted in blue colour).

GRN3 - Traffic Conditions and Anomalous Events

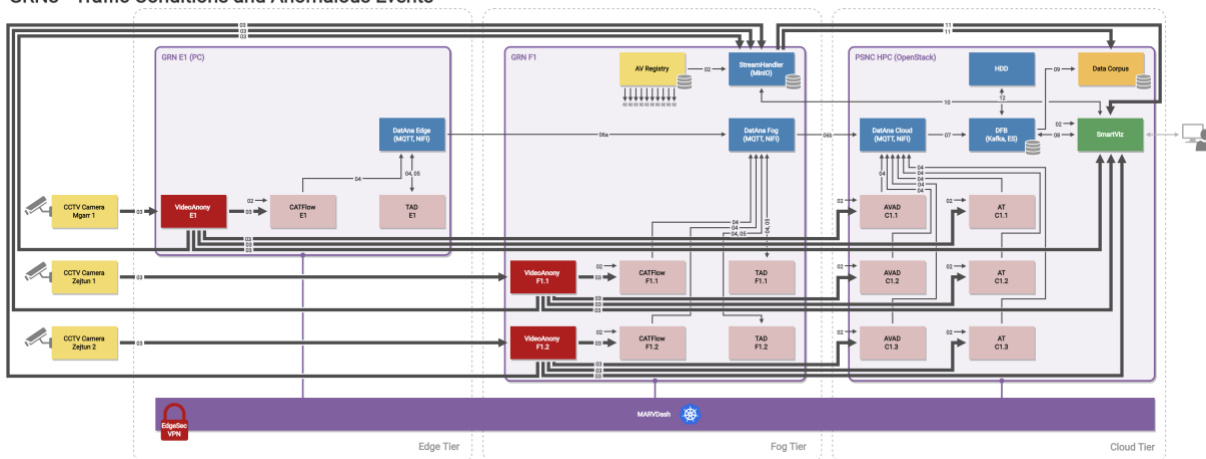


Figure 23: The DMP in the overall system architecture for use case GRN3: Traffic Conditions and Anomalous Events

GRN 4 - Junction Traffic Trajectory Collection

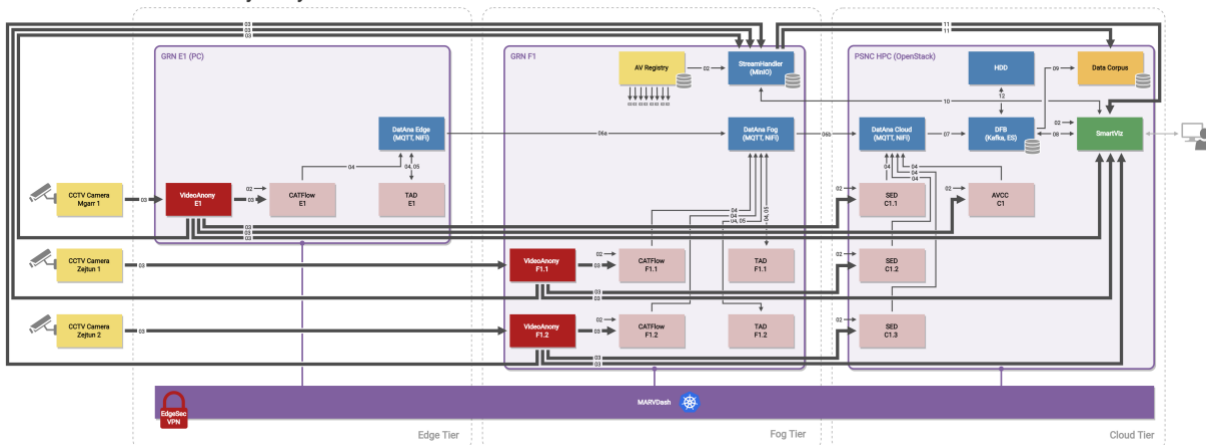


Figure 24: The DMP in the overall system architecture for use case GRN4: Junction Traffic Trajectory Collection

MT1 - Monitoring of Crowded Areas

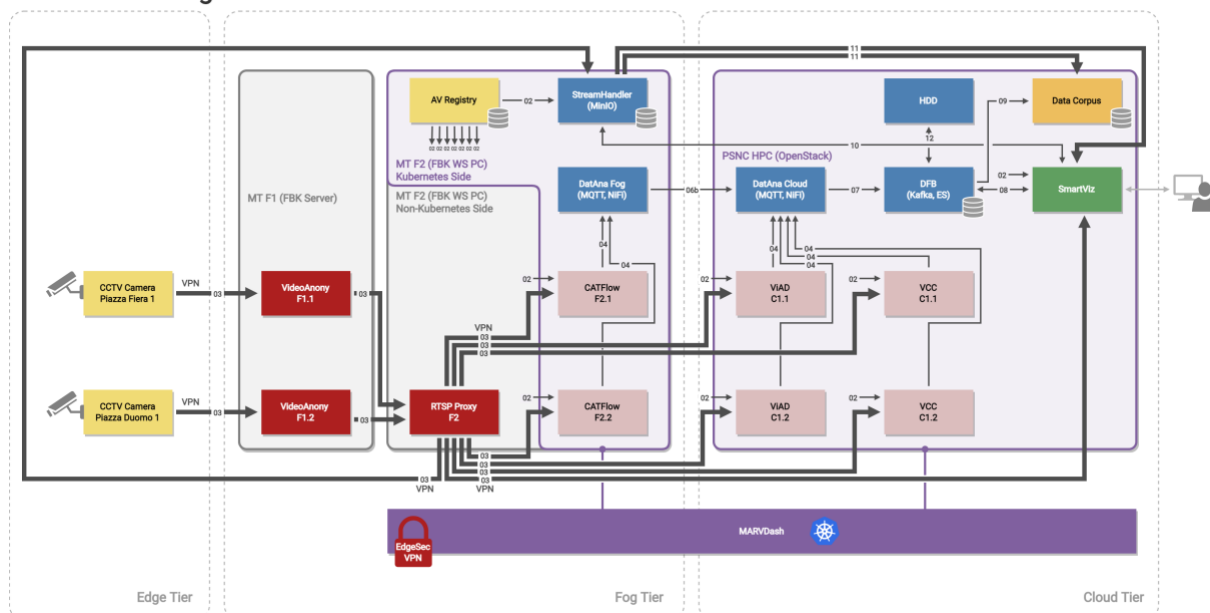


Figure 25: The DMP in the overall system architecture for use case MT1: Monitoring of crowded areas

MT3 - Monitoring of parking places

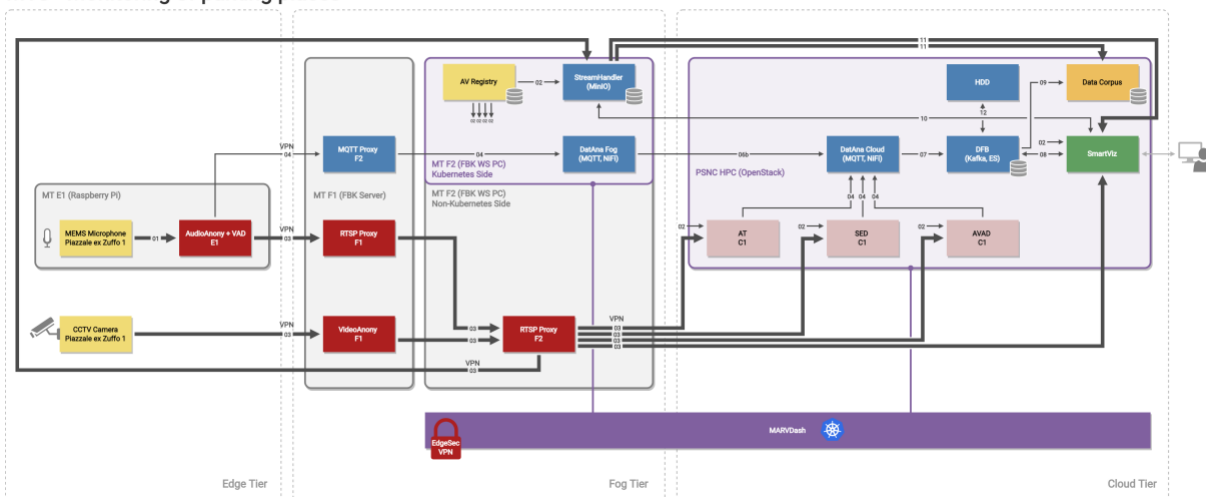


Figure 26: The DMP in the overall system architecture for use case MT3: Monitoring of parking places

UNS1 - Drone Experiment

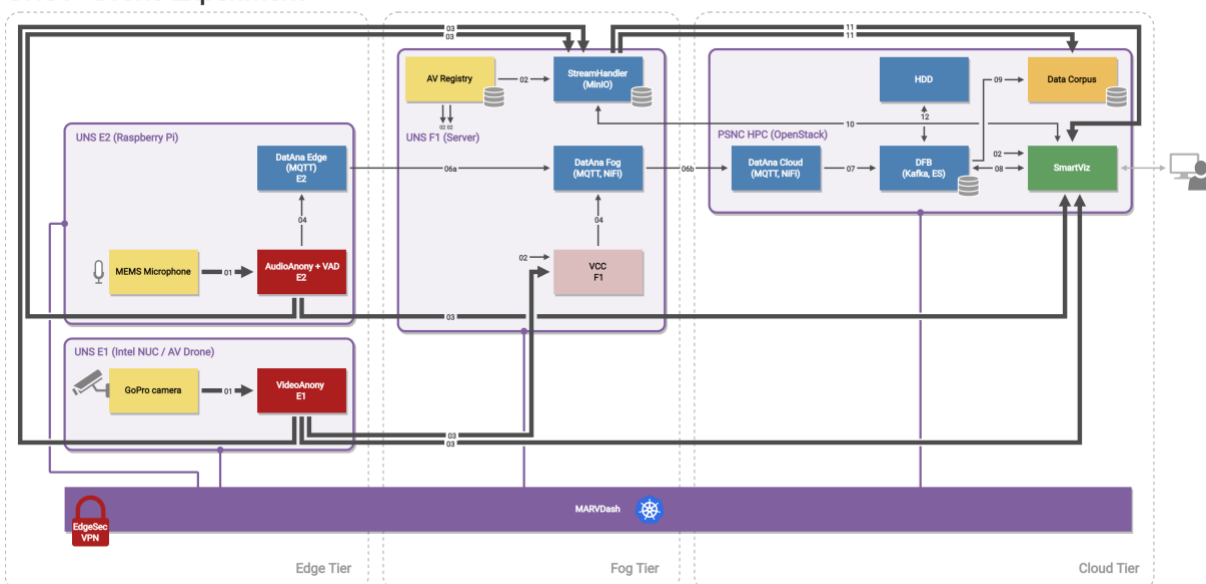


Figure 27: The DMP in the overall system architecture for use case UNS1: Drone Experiment

4.2 The MARVdash facilitator

MARVdash is a dashboard for instantiating services to E2F2C as orchestrated containers. The dashboard aims to make it straightforward for domain experts to interact with resources in the E2F2C MARVEL platform without having to understand lower-level tools and interfaces. MARVdash facilitates the interaction with MARVEL's E2F2C testbed, by supplying the landing page for users working on the platform, allowing them to launch services, design workflows, request resources, and specify other parameters related to execution through a user-friendly interface.

The deployment of MARVdash is based upon a Kubernetes installation as a service. It provides a web-based, graphical interface to manage services or applications that are launched from customisable templates. Additionally, it securely provides multiple services under one externally accessible HTTPS endpoint. A private Docker registry is also available to help organise container images. MARVdash offers high-level user management and isolation of respective services by assigning them to each user's unique namespace. Finally, MARVdash provides the ability to interact with datasets that are automatically attached to service and application containers when launched.

In order for MARVdash to instantiate the E2F2C MARVEL architecture, we use Kubernetes to orchestrate the containers of the different components, with regards to deployment, scaling, and management. All participating servers/nodes in the different layers need to be part of the Kubernetes cluster, however, this is not natively supported by Kubernetes. The main issue is that in most cases the participating servers are behind NAT which is in direct conflict with the fundamental requirements of Kubernetes. To overcome this, it is necessary to bring all servers under the same network subnet and the way we achieve this is by deploying a VPN solution. EdgeSec-VPN is already a component of MARVEL and it is proven an applicable solution for this issue. EdgeSec-VPN will be instantiated in all the participating nodes to enable the process of joining the Kubernetes cluster. The selected solution makes it easy to create virtual networks bypassing intermediate firewalls. The installation of EdgeSec-VPN allows the hosts that join Kubernetes cluster to cross NAT and firewalls and be reachable as if they were under the same network.

MARVdash itself does not do any processing, rather it provides the mechanisms to efficiently deploy the services included in the MARVEL data management toolkit, in all computing continuum layers that support container-based execution. To configure and start services, MARVdash uses a service templating mechanism. Each service is defined with a series of variables and the user can specify values to the variables as execution parameters through the dashboard before deployment. MARVdash will set other "internal" platform configuration values, such as private Docker registry location, external DNS name, and others.

Based on all the above MARVdash will be used to facilitate the realisation of the MARVEL DMP toolkit and the MARVEL Data Corpus.

For the DatAna component (Section 2.1), Apache NiFi is used and the instantiation is realised with the help of Helm²⁹ (Table 7: NiFi). Helm is a Kubernetes deployment tool for automating the creation, packaging, configuration, and deployment of applications and services to Kubernetes clusters. For the Cloud layer, we instantiate a NiFi deployment under a dedicated namespace called *nifi* with registry enabled.

²⁹ <https://helm.sh>

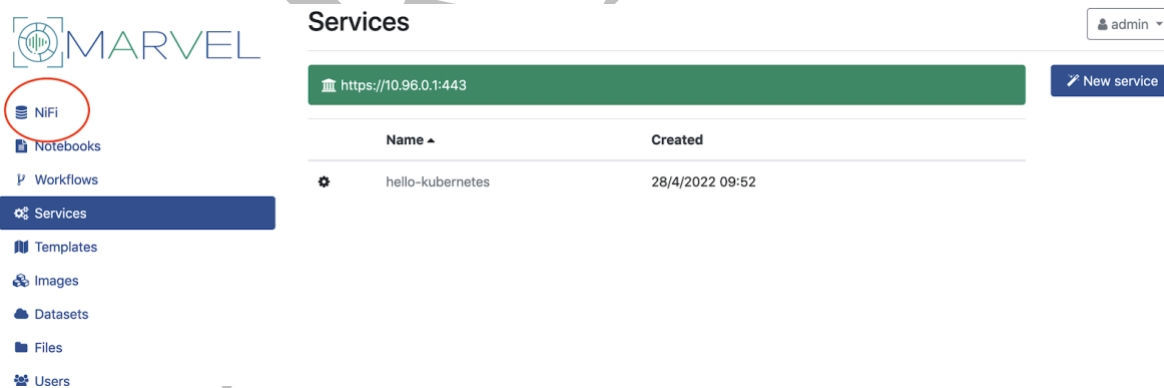
Table 7: NiFi installation with Helm

```

helm install mynifi cetic/nifi --version 1.0.6 \
  --namespace nifi --create-namespace \
  --set registry.enabled=true \
  --set persistence.enabled=true \
  --set service.type=ClusterIP \
  --set ingress.enabled=true \
  --set ingress.hosts\[0\]=nifi.marvel-platform.eu \
  --set auth.oidc.enabled=true \
  --set      auth.oidc.discoveryUrl=https://marvel-platform.eu/oauth/.well-known/openid-
configuration/ \
  --set auth.oidc.clientId=pnARz1f87jebRPWR49tBDgmm0UuZZJ1dVcw3UonA \
  --set
auth.oidc.clientSecret=cWRyDvCZCO3KqDJYHro1kaTHXRTVTdLM8rRKTOKi2R1WMZdmMpQayciwvCU7xVfOgTh1
TXytHpn1drzalUhb38k9hqUcZyMRTP3k0kjkyJ1E7mr7xZkkso0UhJUKc5q \
  --set auth.oidc.claimIdentifyingUser=preferred_username \
  --set auth.oidc.admin=admin \
  --dry-run --debug

```

The above process generates a YAML file which we customised in order to make it applicable for our Kubernetes cluster instance. Additionally, we have configured the NiFi instance to use the OpenID Connect protocol³⁰ which allows the verification of the identity based on the authentication provided by MARVdash's authorisation server in an interoperable and REST-like manner. The NiFi instance is available by clicking on the corresponding menu on the left of MARVdash dashboard (Figure 28).

**Figure 28:** NiFi Cloud Instance available through MARVdash

After clicking on the NiFi link, the user is directed to <https://nifi.marvel-platform.eu/> (Figure 29). If the user has the appropriate rights, he/she will be able to login and use the NiFi Instance.

³⁰ <https://openid.net/connect/>

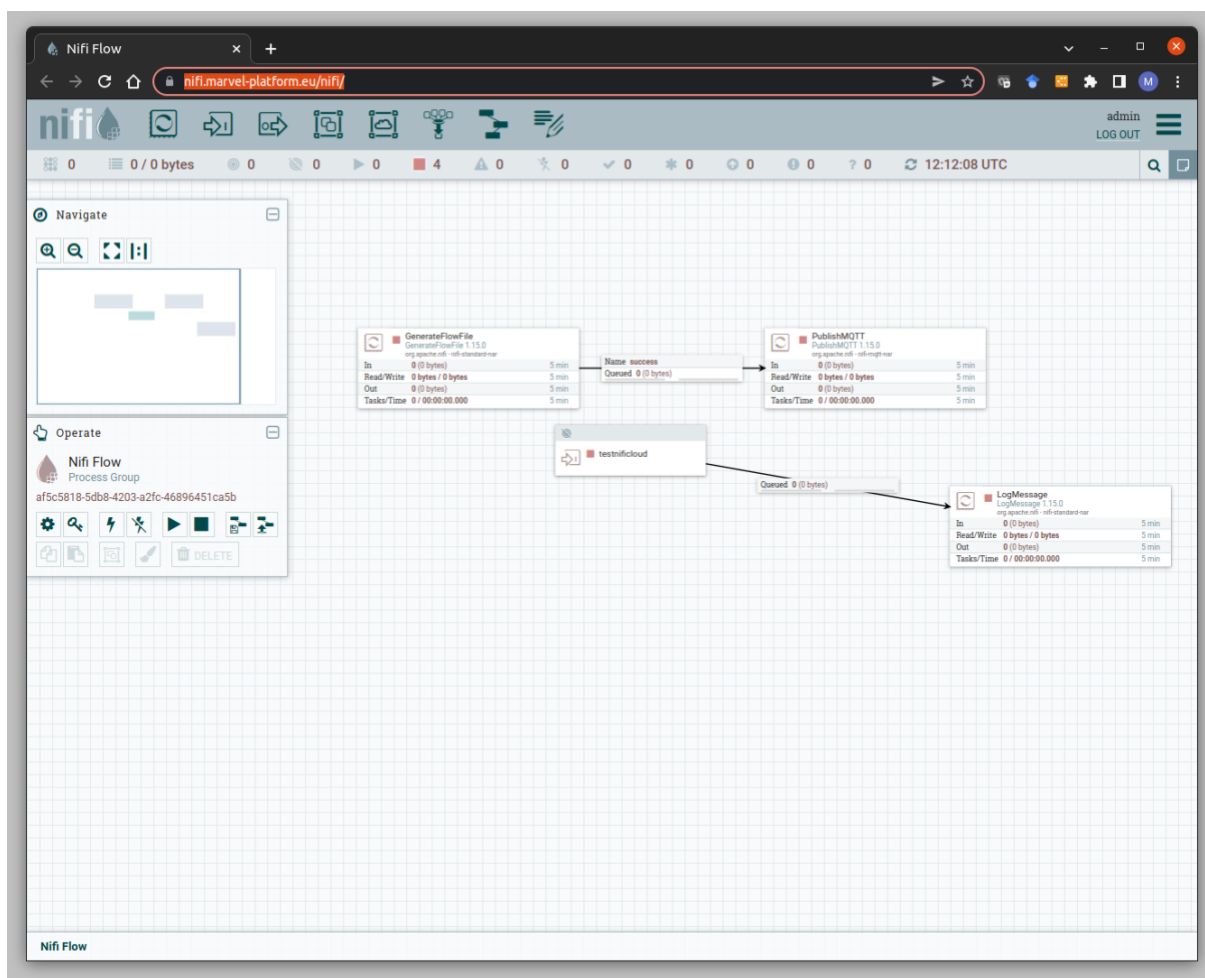


Figure 29: NiFi Cloud instance

The above installation required also a local volume provisioner to create all the persistent volumes that NiFi uses. In order for the above instantiation to work, we installed a local volume provisioner to create all the persistent volumes that NiFi uses.

In addition to the above, a MARVdash-compatible template file was created in Templates where any platform user can instantiate a NiFi service under the respective namespace. The NiFiNew version is an updated version of the old NiFi YAML which launches the NiFi version 1.15.3 (Figure 30).

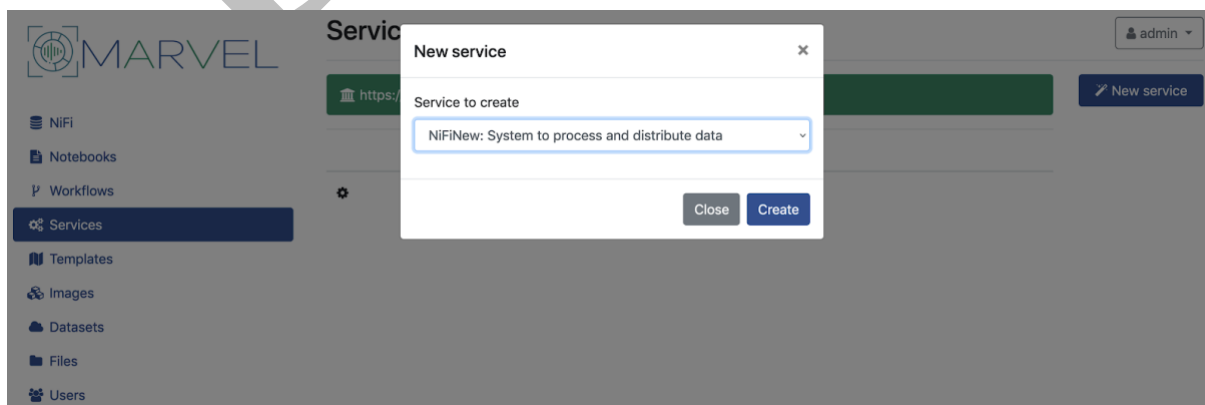


Figure 30: Instantiate NiFi though MARVdash

For the Fog Layer, in the current release of the MARVEL framework, we will customise this YAML appropriately in order to deploy the service in the respective node. For this to happen, we use Kubernetes concepts such as *taints*, *tolerations* as well as *node affinity* (Figure 31). All the layers of the MARVEL architecture are nodes of a Kubernetes cluster and all services are running pods to these nodes.

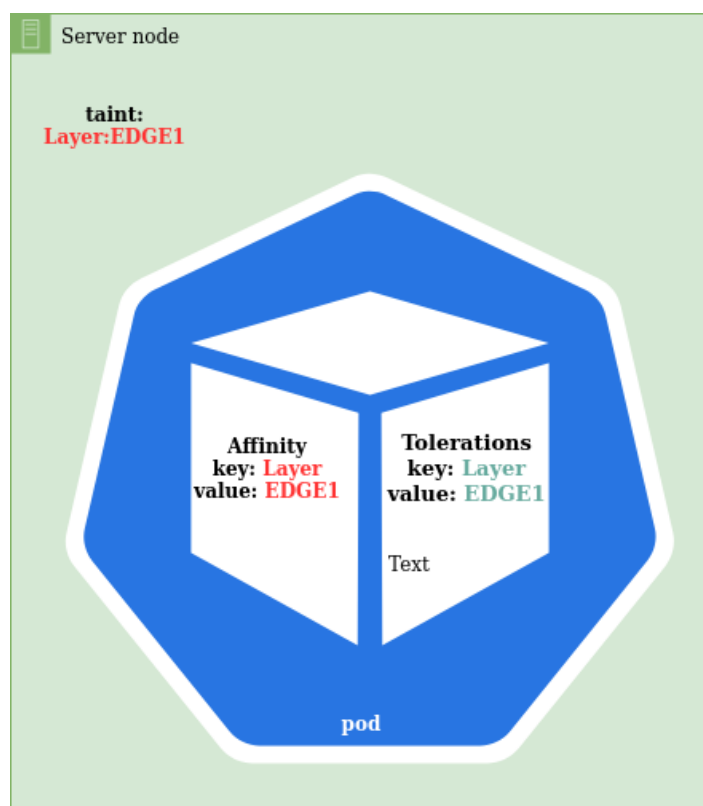


Figure 31: Taints, tolerations, and affinity

Node affinity is a property of Pods that attracts them to a set of nodes either as a preference or a hard requirement. Tolerations are applied to pods, and allow, but do not require, the pods to schedule onto nodes with matching taints. Taints are the opposite since they allow a node to repel a set of pods. Taints and tolerations work together to ensure that pods are not scheduled onto inappropriate nodes. One or more taints are applied to a node; this marks that the node should not accept any pods that do not tolerate the taints.

The command to taint a node is presented in Table 8.

Table 8: Taint nodes command

```
kubectl taint nodes node1 key1=value1:NoSchedule-
```

After we have tainted the node, we can use the tolerations and node affinity in the YAML files (Table 9).

Table 9: Snippet of YAML containing tolerations and node affinity

```
tolerations:
- key: "key1"
  operator: "Equal"
```

```

value: "value1"
effect: "NoSchedule"

affinity:
  nodeAffinity:
    requiredDuringSchedulingIgnoredDuringExecution:
      nodeSelectorTerms:
        - matchExpressions:
            - key: : key1
              operator: In
              values:
                - value1
    
```

For the edge layer as a future work, we will try to instantiate MiNiFi service through MARVdash.

MARVdash also complements the deployment of the Data Corpus (Section 3), by integrating a web proxy service (Figure 32). The proxy implementation allows secure external access to the Data Corpus API services. Additionally, a new ingress endpoint is registered at the Kubernetes side thus providing an HTTPS-compliant URL that ensures the encryption of exchanged data. Finally, the endpoint is configured to employ the authentication mechanism already provided by MARVdash.

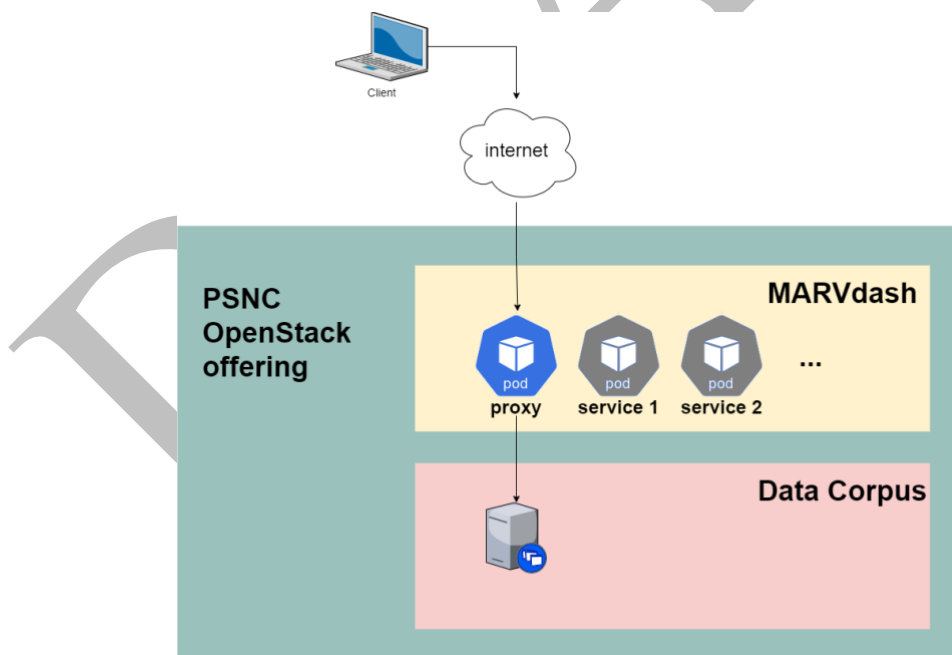


Figure 32: Data Corpus Proxy Logic

5 Future plans towards D2.4 and related KPIs

5.1 DatAna

DatAna will continue evolving in several lines of work in the second half of the project. The main expected enhancements are the following:

- Support for future use cases. DatAna will prepare new data flows for the inference modules to implement new use cases for the pilots and update existing ones as required.
- Deployment of MiNiFi (DatAna Edge) in more devices. For M18 an initial deployment of MiNiFi has been tested using docker containers and MARVDash. However, different versions of docker for MiNiFi in devices with different platforms (i.e., ARM32 or ARM64) have to be provided and tested. The provision of YAML files for the deployment in MARVDash and an easier configuration have to be tested.
- Improvements to the throughput. M18 version is not focusing on optimising the throughput of the system. After a round of benchmarking the idea is to fine-tune DatAna from the configuration (i.e., adjusting configuration properties), scalability (i.e., more nodes or clustering, if required), and implementation of data flows (i.e., by revising potential bottlenecks in the data flows or selecting different data processors for faster processing) perspectives.

DatAna is also involved in several project-related KPIs. The more relevant ones are the following:

- KPI-O1-E1-1 (Different kind of resources to be discoverable: ≥ 3): DatAna is a key component for the inference pipeline data flow for non-AV data acquisition, transformation and transfer from the edge to the fog and cloud. Thanks to the NiFi and the vast amount of off-the-shelf data acquisition processors already in place, DatAna could be also used to ingest data from heterogeneous resources, such as data from sensors, IoT devices, databases, etc., currently not explored in MARVEL. This feature provides extensibility points to the MARVEL framework for future exploitation in environments that require these functionalities.
- KPI-O1-E2-1 (Execution time of data management and distribution improved at least 15%.): This KPI focuses on throughput. Thanks to the underlying Apache NiFi, DatAna can be scaled both horizontally and vertically to ensure the required throughput of the system. NiFi is used in many big data installations around the globe to handle heterogeneous data pipelines. Even if during the scope of MARVEL there is no need to go to a production-grade installation of DatAna, this feature of NiFi offer the possibility to scale MARVEL DMP solutions to the required extent.
- KPI-O1-E2-4 (Improve data distribution in relevant device resource usage at least 15%.): With regards to resource usage, DatAna offers different toolset behind the scenes depending on the nature of the device in which it is installed. For powerful servers or cloud resources, a NiFi cluster can be set and scale up to 1,000 nodes, while for an edge device (i.e., a Raspberry PI), a minimal MiNiFi installation can be adjusted and even downsized to match the existing resources (up to a certain extent). This allows DatAna to be deployed and communicate instances in different environments allowing a seamless data processing and movement as required.

5.2 DFB

In the second half of the project the following activities are being considered for the further development and improvement of the DFB solution within the context of MARVEL:

- Implement a data analytics module that can perform classification and clustering analysis on the inference result data aggregated on the DFB to provide macroscopic insight regarding detected patterns and potential improvements related to the addressed smart city use cases.
- Further refine the REST API for accessing historical data in the Elasticsearch according to the needs of SmartViz.
- Improve the data fusion process of inference results and expose the fusion outcomes to SmartViz and Data Corpus.
- Investigate the distributed deployment of the DFB by adding nodes on both cloud and fog layers.
- Conduct more rigorous testing and measure performance more accurately based on diversified load scenarios.
- Further optimise the DFB performance and particularly the performance of Kafka brokers based on the recommendations from the HDD.

The development of DFB is also related to a number of the project-related KPIs. DFB is expected to play a key role in addressing KPI-O1-E2-3. The KPI refers to the increase in the speed of the data fusion process and is expected to be addressed by improving various performance-related metrics such as latency, data throughput, and response time. To that end, initial measurements were conducted after the MVP release and the plan is to compare against that baseline. Potential improvements are expected through the optimisation of the use of message topics in the DFB Kafka broker. This is to be further assisted through the integration with the HDD component to provide further optimisations and also contribute significantly to KPI-O1-E2-1 (Execution time of data management and distribution improved at least 15%).

5.3 StreamHandler

In the second half of the project the following activities are being considered for the further development and improvement of the StreamHandler solution within the context of MARVEL:

- Further elaborate on the exposed REST API to support more control parameters in the delivery of on-demand AV data streams.
- Consider integrating with the corresponding DatAna instances on the fog layer by subscribing to the respective MQTT topics in order to be immediately informed on any event detection and automatically generate the corresponding compiled AV segment to be accessed by SmartViz.
- Investigate the use of alternative AV data streaming protocols, such as the ONVIF standard, which allows the incorporation of metadata (e.g., time encoding) within the stream.
- Perform exhaustive tests to measure the performance of StreamHandler more accurately and determine its scalability potential.
- Investigate alternative AV data stream segmentation procedures. For example, it will be considered to perform the AV data stream segmentation process according to input from

produced AI inference results in order to maintain and archive only the AV data that correspond to these.

StreamHandler is responsible for the satisfaction of KPI-O1-E2-2, which is associated with increasing the number of different modality data streams. By providing the ability to handle audio-visual data, modalities are clearly increased serving both the project and the component itself that did not have this capacity before. StreamHandler also contributes to KPI-O1-E1-1, KPI-O1-E2-1, KPI-O1-E2-3 and KPI-O1-E2-4.

5.4 HDD

Currently, after taking into account the Apache Kafka basics, HDD provided a modelling approach that considers the most important application constraints and requirements. For the first time in the state-of-the-art, given an Apache Kafka topic, HDD formulated the topic partitioning the problem and we showed its computational intractability, being expressed as an integer program with an objective function of maximising the topic's partitions. HDD provides two heuristic algorithms for addressing the problem and we conducted a performance evaluation against Apache Kafka partitioning configuration recommendations provided by Microsoft and Confluent. We demonstrated via MARVEL-tailored simulations that both the HDD heuristics use the system resources in a more prudent way than the benchmark method, and address well the constraints of replication latency, resource usage, OS load, and unavailability. As future work, we are planning to expand the design and algorithmic solutions to cover large-scale scenarios via simultaneous multi-topic optimisation.

HDD is directly involved in the developments of KPI-O1-E2-1 and KPI-O1-E2-4 which are associated with execution times and resource usage. The metrics that HDD carefully takes into account and considers for optimisation are inherent to the data management processes of Apache Kafka which drives the description of those KPIs. Specifically, the first considers improved execution time of data management and distribution and the second improved data distribution in relevant device resources. As demonstrated in Figure 14, HDD provides the full potential to achieve the envisioned extent of those improvements. HDD is also involved in KPI-O1-E1-1, KPI-O1-E2-3.

5.5 Data Corpus and augmentations

As presented under Section 3, the Data Corpus forms a dataset repository, where pilots can upload datasets which can be shared internally with other MARVEL components or with external users. For the internal operations there are three main interactions: i) internal components can retrieve datasets from the Corpus, ii) data coming from the StreamHandler can be automatically ingested in the Corpus, and iii) inference results coming from DFB can be automatically ingested in existing datasets in the Corpus. Also, augmentation techniques can be performed on the ingested datasets and further facilitate the overall ML procedures. Thereupon, the internal or external users can use a GUI to interact with the Corpus repository (e.g., view datasets, search for data, etc.). Preliminary versions and integrations of all these elements have been accomplished by M18.

Work will continue during the second half of the project, enhancing testing and maturity aspects. Moreover, SLAs will be applied (covering aspects like accessibility, security, and privacy) as well as technical means to assure that their requirements are fulfilled by the deployed system.

The Data Corpus and the augmentation techniques participate in several KPIs and are the main subject of Objective 5 (*Foster the European Data Economy vision and create new scientific and business opportunities by offering the MARVEL Data Corpus as a free service and*

contributing to BDVA standards). However, the main task for the Data Corpus “T2.4 – Sharing multimodal Corpus-as-a-Service: fostering the European data economy vision in smart cities” will start at M20. Therefore, actions toward the fulfilment of the affected KPIs have not been started at the moment (M18).

- KPI-O1-E1-1 (*Different kind of resources to be discoverable: ≥ 3*): The KPI has been fulfilled. The resources that are marshaled for the first phase of the project include: i) cameras, ii) microphones, and iii) drones. Related data has been stored in the Corpus as well.
- KPI-O5-E1-1 (*More than 3.3PB of data made available through a Corpus-as-a-Service*): The KPI is in progress. In the first phase of the project, the main components and their interconnections were developed. A decent amount of data has been currently uploaded in the Corpus, which will be gradually increased during the second period of MARVEL until it reaches the designated threshold.
- KPI-O5-E2-1 (*Release SLAs and consider all the relevant aspects, namely accessibility, operability, managing streaming and network, legal considerations, security, privacy, and technical concerns.*): The KPI is in progress. The main elements will start after M20 and the kick-off of T2.4. Nevertheless, several aspects, such as accessibility and the management of streaming and network, have been already considered for the first integrated prototype of MARVEL.
- KPI-O5-E3-1 (*More than 5 SMEs used the Corpus*): The KPI has not been started. Progress will be recorded in the second half of the project, and probably during the third year of the project when the Corpus SLAs will have been deployed and the overall solution will be more mature in order to be offered to external users.
- KPI-O5-E4-1 (*Maintain the corpus for at least one year after the end of the project*): The KPI has not been started. Nevertheless, PSNC, which provides the infrastructure that is used by the Corpus, is committed to support this KPI after the end of the project.

5.6 Component related KPIs

This Section reports how the presented components involved in T2.2 and T2.3 address the related MARVEL KPIs. The mapping of components to KPIs is consistent with what has been reported in D1.2 and follows the latest advancements of MARVEL. Each KPI is reported with a corresponding set of metrics, the current status and the KPI description.

Table 10: Updated component KPI status since D1.2.

Component	KPI	Metric	Current status	Relevant project KPIs
DFB	Data Integrity	Data loss rate	Data loss rate: 0 (MVP release, as reported in D5.2)	KPI-O1-E2-1 KPI-O1-E2-2 KPI-O1-E2-3
	Scalability	D1.2 status: Hardware resources utilisation, speedup	Will be measured after M18 and in accordance with D5.5 guidelines. The plan is to measure latency and throughput under different load scenarios.	KPI-O1-E2-4
	Availability	Service availability-failed	Service availability:100% Data access restriction:	

		request, data access restriction	None (MVP release, as reported in D5.2)	
	Performance for high volume, heterogeneous data streams	Data transfer latency, data throughput, response time, number of cluster nodes	Data transfer latency 5 ms (200 MB/s load) Data throughput: 605 MB/s Response time: 5 ms (200 MB/s load) Number of cluster nodes:3 (MVP release, as reported in D5.2)	
DatAna	Performance in data rates	Data throughput, response time	Throughput: around 1,1MB/s using files as input. Response time: 67ms Number of cluster nodes:1 (MVP release, as reported in D5.2)	KPI-O1-E2-1 KPI-O1-E2-2
	Scalability	Horizontal scalability in a NiFi cluster, MiNiFi agents handled in multiple devices	Not tested yet So far, no need to use a NiFi cluster to handle the project load	
HDD	Data management and distribution	Data access latency, data loss	Improved data access latency via minimising the Apache Kafka replication latency metric in MARVEL-tailored simulation-based performance evaluation. Improved data loss performance via similarly minimising the Apache Kafka unavailability (Figure 14)	KPI-O1-E2-1
	Device resource usage	Network lifetime, energy consumption rate and energy balance, number of special nodes/equipment	Improved number of special nodes/equipment via minimising the Apache Kafka used brokers metric in MARVEL-tailored simulation-based performance evaluation (Figure 14). This metric indirectly also reflects the energy consumption rate in an Apache Kafka cluster.	KPI-O1-E2-4
StreamHandler	Performance for high volume,	Data transfer latency, data throughput, response time,	Related measurements are expected to take place after M18 and according to the	KPI-O1-E2-3

	heterogeneous data streams	number of cluster nodes	benchmarking plan of T5.4, as reported in D5.4	
Data Corpus	Different kind of resources	T2.4 (not started): Different resources to be discoverable	T2.4 (not started): 3 resource types have been deployed so far: i) cameras, ii) microphones, and iii) drones	KPI-O1-E1-1 KPI-O5-E1-1 KPI-O5-E2-1 KPI-O5-E3-1 KPI-O5-E4-1
	Data made available through a Corpus-as-a-Service	T2.4 (not started): the goal is to ingest 3.3PB of data until the end of the project	T2.4 (not started): Several GBs of data has been ingested so far. This is going to gradually be increased during the second half of the project. MARVEL data corpus GUI and streaming data approach towards the corpus among with targeted augmentation of any ingested data will manage to reach the targeted volume.	
	Release SLAs	T2.4 (not started): definition and deployment of SLAs	T2.4 (not started): STS's Assurance Platform will be considered for the implementation and continuous assurance of the SLAs during the second period of the project	
	SMEs used the Corpus	T2.4 (not started): liaison activities and engagement of external stakeholders	T2.4 (not started): Progress will be recorded at the second half of the project, and probably during the third year of the project, where the Corpus SLAs will have been deployed and the overall solution will be more mature in order to be offered to external users	
	Maintain the corpus after the end of the project	T2.4 (not started): maintenance of the Corpus infrastructure	T2.4 (not started): PSNC, which provides the infrastructure that is used by the Corpus, is committed to support its use after the end of the project	

6 Conclusion

This document provided a description of the components of MARVEL partners during the first half of the project duration (M06-M18) within T2.2 *Data Management and Distribution* and T2.3 *Incremental scheme: continuous augmentation of the dataset*. The developed components focus on four contributions included in the (i) *Data Management and Distribution* and (ii) *Data Corpus-as-a-Service* subsystems, and specifically, DatAna, StreamHandler, DFB, and HDD, as well as Data Corpus. The document targeted the second objective of WP2, i.e., to *collect and analyse the nature and format of experimental data assets and prepare them for processing in the following WPs*. Some of the components presented in this document were integrated in the MARVEL *Minimum Viable Product (MVP)*, as reported in D5.2. Work reported in this deliverable contributes to Objective 1 of the project, i.e., to *leverage innovative technologies for data acquisition, management and distribution to develop a privacy-aware engineering solution for revealing valuable and hidden societal knowledge in a smart city environment*.

The content of the document focused on presenting the design and development of the components provided by the corresponding MARVEL partners within the first 18 months of the project towards achieving the project objectives. The final version of the management and distribution toolkit in the MARVEL project will be documented near the end of the project (M30) in D2.4 and will contain the enriched set of components further improving the set of data-related capabilities in MARVEL. The future plans of the work reported involve (i) the further development and enhancement of the presented components towards a more consolidated and holistic version, (ii) the improved efficiency when targeting the base KPIs of interest, especially when it comes to metrics such as latency, throughput, resource utilisation and scalability, and, (iii) addressing the final set of KPIs which are related primarily to T2.4.

7 References

- [1] MARVEL, “D1.3: Architecture definition for MARVEL framework,” 2020. [Online]. Available: <https://doi.org/10.5281/zenodo.5463897>.
- [2] MARVEL, “D5.1: MARVEL Minimum Viable Product,” 2021. [Online]. Available: <https://doi.org/10.5281/zenodo.5833310>.
- [3] T. P. Raptis and A. Passarella, “On Efficiently Partitioning a Topic in Apache Kafka,” in *International Conference on Computer, Information, and Telecommunication Systems*, 2022.
- [4] P. Dobbelaere and K. Sheykh Esmaili, “Kafka versus RabbitMQ: A comparative study of two industry reference publish/subscribe implementations: Industry Paper,” in *11th ACM International Conference on Distributed and Event-based Systems*, 2017.
- [5] G. M. D'silva, A. Khan, Gaurav and S. Bari, “Real-time processing of IoT events with historic data using Apache Kafka and Apache Spark with dashing framework,” in *2nd IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology*, 2017.
- [6] J. Andersson, J. J. Alonso Moya and U. Schwickerath, “Anomaly Detection for the Centralised Elasticsearch Service at CERN,” *Frontiers in Big Data*, vol. 4, 2021.
- [7] T. P. Raptis, A. Passarella and M. Conti, “Distributed Data Access in Industrial Edge Networks,” *IEEE Journal on Selected Areas in Communications*, vol. 38, pp. 915-927, 2020.
- [8] T. P. Raptis, A. Passarella and M. Conti, “Energy efficient network path reconfiguration for industrial field data,” *Computer Communications*, vol. 158, pp. 1-9, 2020.
- [9] J. Xu, J. Yin, H. Zhu and L. Xiao, “Modeling and verifying producer- consumer communication in kafka using csp,” in *7th Conference on the Engineering of Computer Based Systems*, 2021.
- [10] M. Tsenos, N. Zacheilas and V. Kalogeraki, “Dynamic rate control in the kafka system,” in *24th Pan-Hellenic Conference on Informatics*, 2020.
- [11] T. Aung, H. Y. Min and A. H. Maw, “Enhancement of fault tolerance in kafka pipeline architecture,” in *11th International Conference on Advances in Information Technology*, 2020.
- [12] H. Wu, Z. Shang and K. Wolter, “Performance prediction for the apache kafka messaging system,” in *IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems*, 2019.
- [13] X.-C. Chai and etal., “Research on a distributed processing model based on kafka for large-scale seismic waveform data,” *IEEE Access*, vol. 8, pp. 39971-39981, 2020.
- [14] S. Langhi and R. Tommasini, “Extending kafka streams for complex event recognition,” in *IEEE International Conference on Big Data*, 2020.

- [15] M. Gutlein and A. Djanatliev, “On-demand simulation of future mobility based on apache kafka,” in *Simulation and Modeling Methodologies, Technologies and Applications*, Springer International Publishing, 2022, pp. 18-41.
- [16] M. H. Javed, X. Lu and D. K. D. Panda, “Characterization of big data stream processing pipeline: A case study using flink and kafka,” in *Fourth IEEE/ACM International Conference on Big Data Computing, Applications and Technologies*, 2017.

DRAFT