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

D2.4: Management and distribution Toolkit – final version[†]

Abstract: This document describes the final version of the MARVEL project's management and distribution toolkit. In this document, an in-depth presentation of the components responsible for the data management and distribution in terms of novelty, design, implementation, connection with other components, how they are used within the Pilots, and their performance performed. The components under the management and distribution toolkit are ATOS's DatAna, ITML's DFB, INTRA's StreamHandler, and CNR's HDD. Moreover, the contribution of the management and distribution tools in the Edge-to-Fog-Cloud framework and their future directions in the final part and after the MARVEL project are presented.

Contractual Date of Delivery	30/06/2023
Actual Date of Delivery	29/06/2023
Deliverable Security Class	Public
Editor	*Ilias Seitanidis (INTRA)*
Contributors	CNR, ITML, INTRA, ATOS, GRN, UNS, FORTH
Quality Assurance	*Grigorios Kalogiannis (STS)* *Alessio Brutti (FBK)*

[†] *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
12	COMUNE DI TRENTO	MT	Principal Contractor	IT
13	UNIVERZITET U NOVOM SADU FAKULTET TEHNIČKIH 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. *Grigorios Kalogiannis (STS)*
2. *Alessio Brutti (FBK)*

Revisions

Version	Date	By	Overview
0.1.4	29/06/2023	INTRA	Final version
0.1.3	20/06/2023	INTRA	Second draft version
0.1.2	13/06/2023	INTRA	First draft version
0.1.1	03/05/2023	INTRA	Second version of ToC
0.1.0	21/04/2023	INTRA	Initial 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.

Table of Contents

List of Tables

List of Figures

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
R1	Release 1 of the MARVEL framework
R2	Release 2 of the MARVEL framework

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

Executive Summary

The present document serves as a follow-up to D2.2 and focuses on the development and enhancements of the Management and Distribution Toolkit. This toolkit aims to address the problem of AV data handling within the MARVEL project. The purpose of this deliverable is to report the second and last version of the management and distribution toolkit encompassing all the updates from M18 till M30 of the project, in addition to the initial implementation that was reported in D2.2 (M18). The work presented here is the outcome of Task 2.2 - "Management and distribution toolkit -initial version". The content of this document focuses on the updates carried out in several sectors with design, implementation, and performance being some of the main points of interest. The tools' description presented here regards the R2 (Release 2 of the MARVEL framework) deployment phase which constitutes the final version of the MARVEL platform.

Deliverable D2.4 represents a significant milestone in the development and evolution of the MARVEL Data Management Toolkit. The enhancements made through this follow-up build upon the foundation laid in D2.2, providing a powerful and comprehensive solution for effective AV data management. Notable achievements include optimisation tweaks in DatAna and Data Fusion Bus (DFB) components, performing extensive research and more concrete results for the evaluation experiments of the Hierarchical Data Distribution (HDD) component, and the restructure of the StreamHandler component. Another key mile[illegible]e presented in this document is the increase in the number of deployments perfor[illegible]ough the MARVEL project, reaching a total of ten in the period between M1[illegible]d M30 of th[illegible]project.

1 Introduction

1.1 Purpose and scope

Deliverable D2.4 entitled "Management and Distribution toolkit – final version", reports on the progress made through the relative activities carried out within the T2.2 framework for the period from M19 (July 2022) till M30 (June 2023). The three key areas that MARVEL's Data Management Toolkit (DMT) focuses on are a) data management, b) data storage, and c) data distribution. In D2.2[1] a primitive analysis of these tools was reported while in D2.4 the final version of the DMT components with optimisations in performance and/or efficiency is described. These optimisations lead to the fulfilment of the MARVEL's objectives, such as Objective 1: 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.

1.2 Contribution to WP2 and project objectives

The work presented in this deliverable was conducted under the T2.2 framework and some parts of this work are reported in other WP deliverables as well. The work reported in the following Sections regards the updates of the tools after D2.2. Specifically, the data flow, distribution, management, and storing across the Edge-to-Fog-to-Cloud (*E2F2C*) continuum, considering all the pilot requirements and constraints that came up since M18. Further automations and improvements on the efficiency of the previously reported management solutions have been achieved, with a continuous development and integration process, in collaboration also with the activities of WP3 and WP5, just to name a few. In addition, D2.4 is strongly coupled with the Pilots' activities. During the course of the project, the components presented in this document fulfilled their initial goals, defined through the project's KPIs. More details regarding the components' KPIs and metrics can be found in WP5 and WP6 deliverables D5.2[2], D5.5[3], and D6.2[4]. Some of the project's KPIs that the DMT components achieved are:

- KPI-O1-E2-1: Execution time of data management and distribution improved at least 15%.
- KPI-O1-E2-2: Increase the number of different modality data streams that can be handled by 30%.
- KPI-O1-E2-3: Increase the speed of the fusion process by at least 20%
- KPI-O1-E2-4: Improve data distribution in relevant device resource usage at least 15%.

[1] MARVEL D2.2: Management and distribution Toolkit – initial version, 2022. https://doi.org/ 10.5281/zenodo.6821195

[2] MARVEL D5.2: Technical evaluation and progress against benchmarks – initial version, 2022. https://doi.org/10.5281/zenodo.6322699

[3] MARVEL D5.5: Technical evaluation and progress against benchmarks. *To appear.*

[4] MARVEL D6.2: Evaluation report, 2023. https://doi.org/10.5281/zenodo.7296312

Table 1: Project's KPIs to DMT components mapping

Project's KPI	Components
KPI-O1-E2-1	DatAna, DFB, HDD
KPI-O1-E2-2	DatAna, DFB
KPI-O1-E2-3	DFB, StreamHandler
KPI-O1-E2-4	DFB, HDD

1.3 Relation to other WPs, deliverables, and activities

In WP2, the work carried out in T2.2 aims to maximise the impact of the MARVEL Data Corpus on the international scientific and research community regarding data management and distribution for smart cities. This involves utilising the components from both T2.2 and T2.3. Additionally, the work in T2.2 is interconnected with various tasks in other WPs.

Specifically, it is closely related to T1.3, which deals with the experimental protocol and real-life societal trial cases in smart city environments as outlined in D1.2. The management and distribution functionalities in T2.2, along with the corresponding data-oriented components, are described in T1.4. This work refines the specification of the conceptual architecture of the MARVEL E2F2C ubiquitous computing framework as documented in D1.3[5].

The management and distribution functionalities developed in T2.2, using the components discussed in the subsequent sections, are continuously integrated into the overall MARVEL architecture. This integration takes place in WP5, specifically in T5.2, which focuses on resource management and optimised automated use case, and T5.3, which deals with continuous integration towards the realisation of the MARVEL framework.

The effectiveness of the proposed components will be assessed in T5.4 against societal, academic, and industrial validated benchmarks, and in T5.5 as the prototype progresses towards the final solution. Two components, namely DatAna and DFB, were included in the Minimum Viable Product (MVP), as reported in D5.1[6]. Furthermore, the initial release of the MARVEL integrated framework (D5.4[7]) included all four components: DatAna, DFB, StreamHandler, and HDD.

1.4 Status updates since D2.2

In D2.2, the initial state (up to M18 of the project) of the DMT was described. Since then, the partners have performed several modifications to their components. DatAna and DFB focused on the adaptation of more use cases while maintaining the same level of performance. StreamHandler received an architectural and functional update. CNR expanded their experiments for the HDD to obtain more concrete results as well as to enhance the performance within the MARVEL project. The progress demonstrated in the following sections, either minor or major, contributes to the overall enhanced performance of the MARVEL framework.

[5] MARVEL D1.3: Architecture definition for MARVEL framework, 2020. https://doi.org/10.5281/zenodo.5463897

[6] MARVEL D5.1: MARVEL Minimum Viable Product, 2021. https://doi.org/10.5281/zenodo.5833310

[7] MARVEL D5.4: MARVEL Integrated framework – initial version, 2022. Confidential

1.5 Structure of the document

The document is structured as follows. Section 1 describes the purpose and scope of this document, the contributions of the work conducted in T2.2 to WP2 and to the project objectives, and its relation to other WPs and deliverables. Section 2 offers a detailed description of the updates carried out since D2.2 for the components of the distribution and management toolkit in terms of design, implementation, and performance, demonstration and interaction with other components. Section 3 presents the components' position within the Edge-to-Fog-Cloud continuum. Finally, Section 4 outlines the future plans, for further development of the components beyond M30.

2 Components of the DMP toolkit

2.1 DatAna

As explained in deliverable D2.2, the main role of DatAna in MARVEL is related to gathering the outputs of all inference models in real-time from different edge, fog, and cloud nodes. DatAna transforms these outputs into the MARVEL-compliant data models and sends these results to the DFB at the cloud for further processing, fusion, and storage. DatAna can be used for both streaming and batch data acquisition. For R1 and R2, all the operations are done in real-time, allowing gathering and processing the results of the inference pipelines introducing a minimal latency, thus contributing to the quick reaction of the system to anomalies and events. This is possible due to the fact that DatAna is based on the Apache NiFi[8] ecosystem, a very scalable, low-latency set of tools with a proven track record. 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.

2.1.1 Progress beyond the state-of-the-art

Section 2.1.1 of D2.2 reported on the progress beyond the state-of-the-art of DatAna. Since then, no major changes have arisen, as the work in DatAna in the second half of the project has been focused on serving the new R2 use cases and fine-tuning the work done for R1. In summary, the main progress highlighted in D2.2 is related to the usage of DatAna as an inference data ingestion, transformation, and movement through the different layers of the computing continuum in a reliable, efficient, secure and scalable fashion, thanks to the underlying capabilities of NiFi. Data is gathered in each of the elements of the infrastructure by a dedicated combination of MQTT[9] brokers with Apache NiFi services are located in the nodes and moved up to the cloud seamlessly.

2.1.2 Design methodology

The design methodology of DatAna was introduced in section 2.1.2 of D2.2. The Apache NiFi main elements (flowfile, processors, connectors and NiFi) are described in that section as a context to better understand the functioning of DatAna.

In general, the design of DatAna has not changed since then. The main idea, introduced in the previous subsection, is the deployment of a set of NiFi and MQTT services in the majority of the infrastructure nodes located at the edge, fog and cloud nodes used in MARVEL for the 3 pilots and the 10 use cases, connected as a tree or topology. The final view of this tree used in R2 can be seen in Figure 1.

[8] https://nifi.apache.org/

[9] https://mosquitto.org/

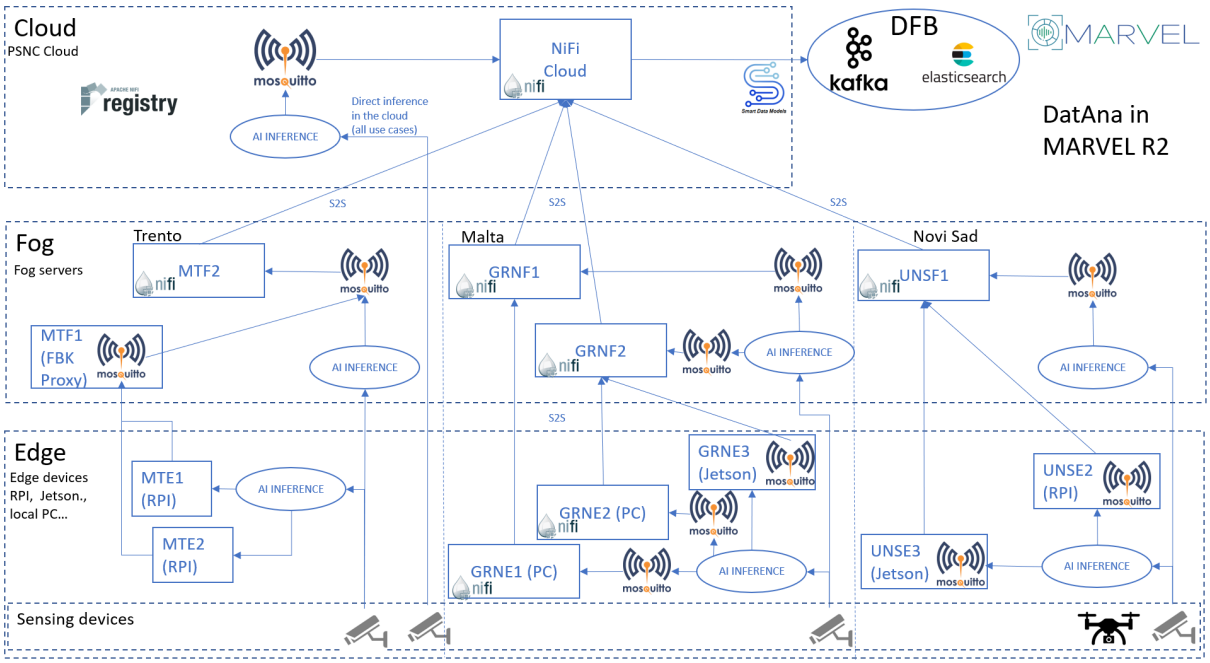


Figure 1. DatAna topologies for R2

The main difference between the equivalent Figure shown in D2.2 and Figure 1 is that in the later DatAna is particularised to the use cases in the R2 release, showing only the DatAna services used in each of the nodes for these specific scenarios. In general the design of the topology requires an MQTT broker in the nearest node to where the inference models run. Their output is directed as messages to a set of dedicated MQTT queues named after the component according to the MARVEL formatting guidelines (e.g., "sed" queue for the SED component). DatAna then provides in the dedicated NiFi node a set of dataflows that can subscribe in real time to the different MQTT topics, take the messages and transform them to the MARVEL data models for Alert, Anomaly or MediaEvent (extended from their equivalent data modes from the Smart Data Models initiative to allow further standardisation of our results). The results are directed up in the topology to the next NiFi node via the NiFi Site-to-Site (S2S) protocol up to the cloud. Then the correctly formatted messages are sent to the DFB Kafka for further processing.

The Apache NiFi Registry[10] located at the cloud, provides dataflows version control and has been extensively used in R2, while in R1 was only used sparsely and as a proof-of-concept. This component is key to not duplicating pieces of code or dataflows, especially useful to deploy similar dataflows in different NiFi nodes (e.g., the dataflows in NiFi related to the transformation of the VAD component placed in different layers and devices).

The equivalent section of D2.2 provides a set of configuration and design principles for NiFi pipelines and dataflows. These have not changed. Please, refer to section 2.1.2 of D2.2 for the details.

[10] https://nifi.apache.org/registry.html

2.1.3 Implementation

As in the previous sections, the implementation of DatAna for R2 follows a very similar approach to what was described for R1 in section 2.1.3 of D2.2. As a summary, we have provided and implemented the following elements:

- We have provided the NiFi containers (v1.15.3, for compatibility with the R1 version) and a Helm Chart installation approach for NiFi based on the cetic[11] helm chart to be adapted to deploy NiFi as a service on a Kubernetes cluster. FORTH proceed to use and fine-tune the template to enable it in the different layers in MARVdash. NiFi and MQTT are deployed as shown in Figure 1.
- The security in the communications enabled via NiFi S2S protocol and TLS (by populating the necessary keystores and trust stores) has been set. All the instructions on how to enable the security and the configuration have been provided in the MARVEL GitHub for DatAna[12].
- NiFi dataflows to enable the ingestion have been developed for all the inference components at play, namely AAC, AT, CATFlow-P, CATFlow-V, RBAD, SED, SELD, VAD, VCC, AVCC, ViAD, AVAD, and YOLO-SED. Figure 2 shows an example of one of these dataflows developed for the new components in R2.

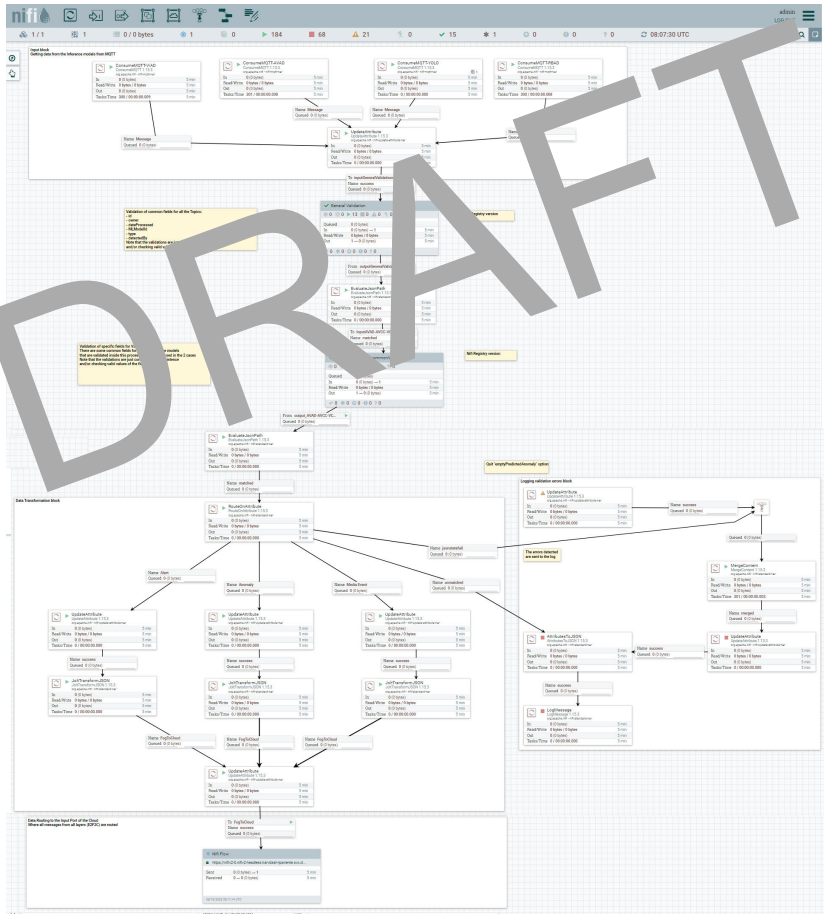


Figure 2. Example dataflow in NiFi for R2

[11] https://github.com/cetic/helm-nifi

[12] https://git.marvel-project.eu/marvel/datana/datana/-/tree/main/docker/secureNifi

Figure 2 shows a NiFi dataflow that serves to transform the outputs of the AVAD, ViAD, RBAD, and YOLO-SED components. These outputs are similar; therefore, a single dataflow is enough for transforming them to the Anomaly, Alert or MediaEvent data models. The dataflow is structured in a way that first subscribes from the respective MQTT queues (in this case, "avad", "viad", "rbad" and "yolo-sed" queues) located in the node. Then a couple of blocks of the dataflow validate that the format of the data received is the expected (if not, an error is raised and directed to the log while the message doesn't go forward). After that, the transformations to the "Alert", "Anomaly" or "MediaEvent" data models are done in the following block, before sending the results to the upper NiFi layer in the topology (the last processor at the bottom of the figure). All the rest of the dataflows for inference data transformation work in the same way as the one described above.

As a side note, some specific nodes at the edge do not have NiFi services, but just MQTT as it can be perceived in Figure 1. This is due to the implementation decision to not overload some edge devices with NiFi or Apache MiNiFi[13] services, since these nodes need to run already several services (anonymisation, streaming, inference). Although the MiNiFi footprint is not too high, it may impact negatively on the performance. Therefore, in some cases it has been decided to deploy only an MQTT service at the edge, and the next NiFi in the topology (at the fog) would subscribe to the messages produced at the MQTT at the edge for further processing. The result is more performant, provides the same functionality and allows running more services, or the same with better performance, in the edge devices.

2.1.4 Connection to other components

As introduced in D2.2 section 2.1.4 and commented in previous subsections, DatAna has connections with all the inference models via dedicated MQTT queues in each of the layers of the E2F2C continuum. This connection via MQTT allows the separation of the components and DatAna. This makes for a loosely coupled integration of the inference pipeline.

Once the data arrives at the upper NiFi in the topology (the NiFi cloud) via a NiFi input port, the data is directed to the DFB via equivalent Kafka topics to the MQTT queues, as shown in Figure 3.

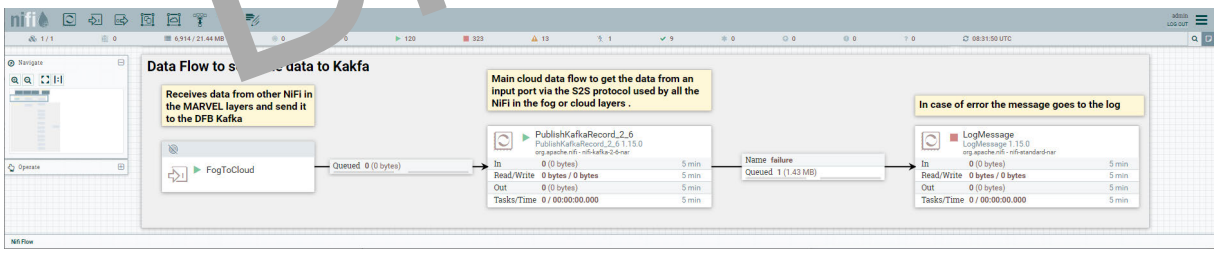


Figure 3. Sending the inference data to the Kafka of the DFB from DatAna cloud

This last action allows real-time streaming of the inference pipeline results and decouples the interactions between DatAna and the DFB via the Kafka pub-sub mechanism.

2.1.5 Demonstration

As stated previously, NiFi and MQTT have been included in most of the nodes at edge, fog, and cloud across the MARVEL pilots as needed. NiFi has been deployed as a service in the Kubernetes cluster, with the required YAML files, in at least 7 locations. One at the cloud (master instance of the topology), at the 4 fog servers (one for UNS, one for MT and two in

[13] https://nifi.apache.org/minifi/

GRN), and at least in 2 edge devices (2 in GRN). MQTT instances associated with DatAna have been deployed in the same locations, and additionally in all the rest of the edge devices. One NiFi Registry has been deployed at the cloud and contains versions of the dataflows serving all the rest of the NiFi in the topology.

All the necessary dataflows for the inference components running at the specific node have been deployed in the correspondent NiFi, according to the architecture of the use cases. Therefore, each NiFi instance only contains the dataflows needed for the specific use cases that the node is working for. All these dataflows are using the latest versions stored in the NiFi Registry.

Figure 4 shows an example of a message processed by DatAna from VCC before sending it to the DFB. The resulting message shows the required field by the MediaEvent data model (field type). Note for instance that this message was produced at the cloud (detectedBy=”PSNC-C1”), by a specific instance of the VCC model (as in the MLModelId field), from a video stream coming from a specific camera from the UNS setting (content of CameraId field), for a UNS use case (owner=”UNS1”). The model predicts in this case 3 elements (data.predictedCount=3) in the video at the specific time (timestamp from May 9 2023). Some other fields provide extra info about the processing time (dateCreated and dateProcessed), which in combination with other fields to be filled in by the DFB (dateModified and dateStored) will serve later on to understand the overall latency introduced by the system from a benchmarking perspective. DatAna has produced all these fields using transformation rules embedded in the dataflows, in this case in the dataflow for VCC executed at the cloud NiFi instance.

Figure 4. Sample data from VCC inference results after DatAna process it as MediaEvent

As occurs with the rest of the DMP platforms, DatAna works behind the scenes. Therefore, its demonstration is only visible from the user point of view if the messages arrive to the SmartViz tool correctly after being stored by the DFB. The NiFi GUI allow nevertheless to check the messages that are passing through NiFi. Using the latter interface (only available for developers or admin users), it is possible to see the messages as they pass through the different steps to the

dataflow, as shown for instance in Figure 4. NiFi allows to track the lineage of the data as it passes to the different steps of the dataflow.

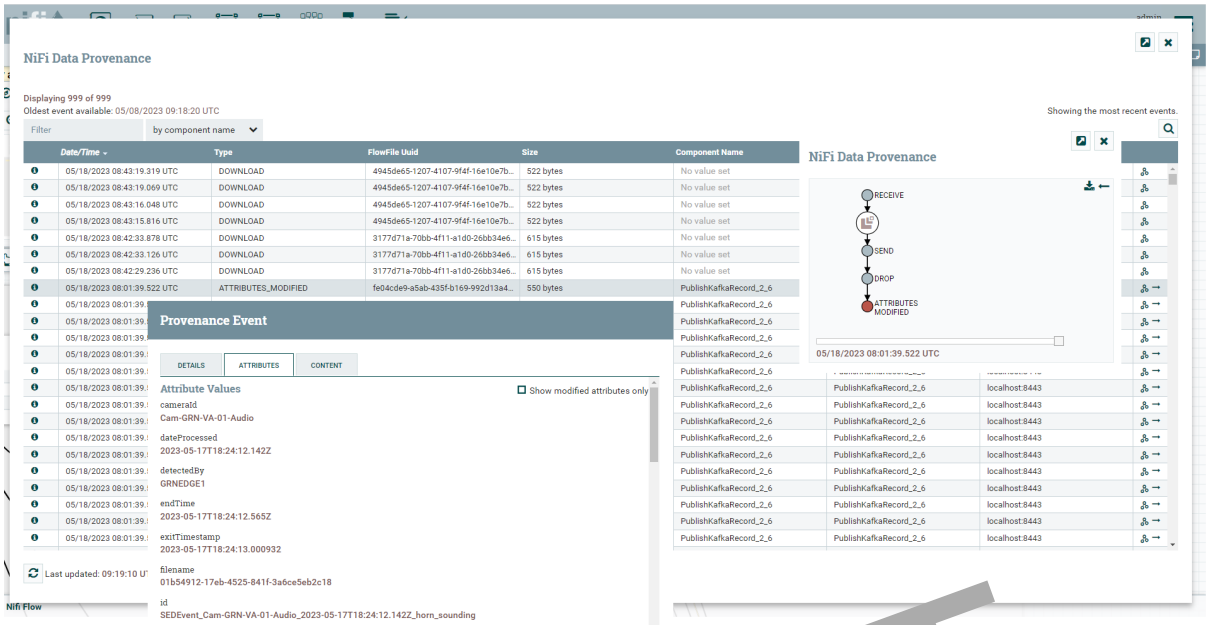


Figure 5. Sample of data provenance inspection in NiFi of message from a SED inference results

Finally, NiFi also provides the means to check many other aspects of its utilization. For instance, admins might be interested in checking the average number of processes running during a time interval. The GUI provides the possibility to check among many other parameters of the system, as shown in Figure 6.

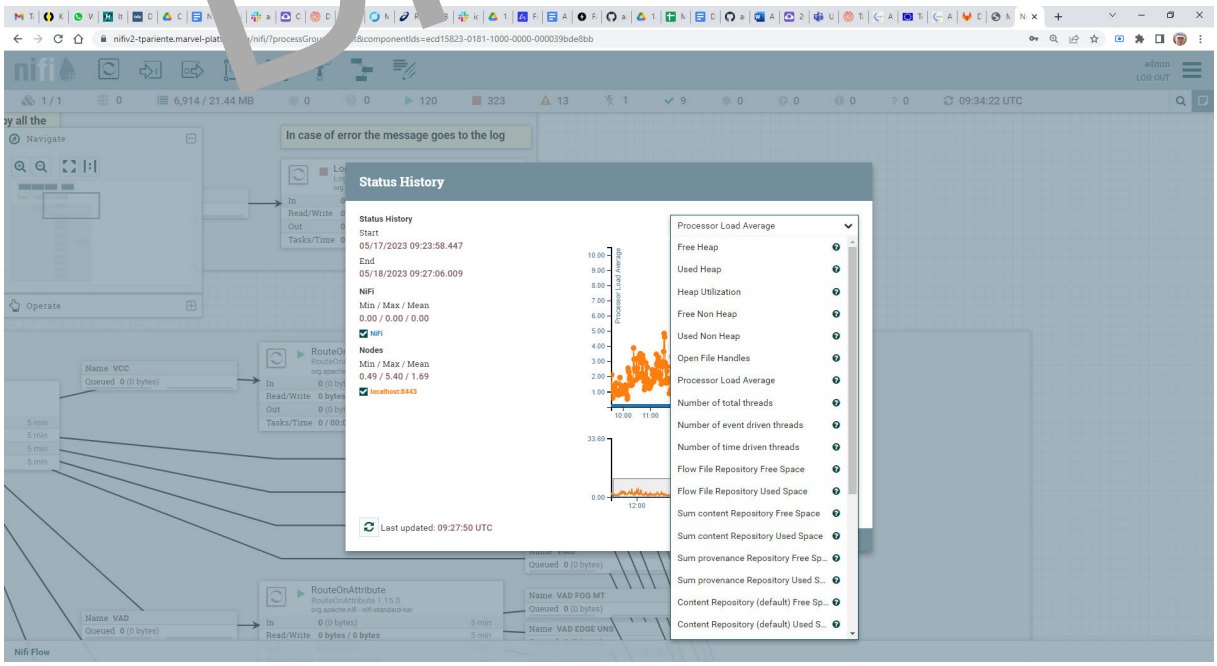


Figure 6: NiFi admin interface to check status of the system

2.1.6 Performance

As stated in D2.2, the main metrics related to DatAna are throughput and latency. In the scope of the benchmarking activities carried out in WP5, some measurements of these metrics have been provided in the first quarter of 2023. It is important to consider that DatAna as a whole is composed of a topology of multiple components residing in different layers. Therefore, its latency is the combined latency since the inference models put their output in MQTT until the processed message arrives at the DFB. This means the latency combined of MQTT, plus the edge, fog, and cloud NiFi that intervene in each of the cases, plus the latency introduced by the network in the communication among these layers up to the DFB. Nevertheless, in the benchmarking results, the average latency introduced by DatAna is between 100 and 400ms, below or comparable in most of the cases to the latency introduced by the inference components. A more detailed performance evaluation will be carried out in the scope of WP5.

It is important to point out that DatAna is operating in all nodes with a standalone NiFi instance with minimal settings, which are enough to operate within the current MARVEL infrastructure and needs. If necessary, especially at the cloud, NiFi could be deployed as a cluster. This will provide the necessary scalability for more demanding environments. For a more in-depth analysis of performance metrics, Cloudera produced a benchmarking study[14] reporting the scalability and performance (data rates) using very demanding workloads.

2.2 Data Fusion Bus (DFB)

As reported in D2.2, the DFB is a customisable component that implements a trustworthy way of transferring large volumes of heterogeneous data between several connected components and a permanent storage. It comprises a collection of dockerised, open source components which allow easy deployment and configuration as needed.

The DFB aggregates data that originates from multiple sources and is relayed by DatAna as streams. The DFB permanently stores this data and exposes it to SmartViz both in real-time and asynchronously in the form of historical data. The DFB is positioned at the cloud layer and considered an integral part of the DMP and inference pipeline, with a pivotal role in all 10 R2 use cases.

The DFB can also be used for interfacing with third-party components that are not part of the MARVEL framework but may need to act as data producers or consumers in potential future use cases (e.g., other AI components, data visualisation tools, etc).

2.2.1 Progress beyond the state-of-the-art

The main novelties and innovations of DFB were documented in detail in section 2.2.1 of D2.2, and are associated with the seamless integration of underlying state-of-the-art technologies (e.g., Apache Kafka, Elastic Search, Grafana) and their efficient coupling with innovative custom-built services in order to deliver a comprehensive, customisable, scalable, and extensible data management solution that is optimised for IoT and Big Data Analytics applications, offering high data reliability, insightful data visualisations, secure data access, and effective data transformations. At the same time, DFB provides all the necessary endpoints to connect with components of the MARVEL framework, while the interface of the DFB with the HDD component further reinforces and optimises the scalability potential of the DFB. The novelties and innovations reported in D2.2 are still relevant as no major structural changes have been introduced to the DFB for R2.

[14] https://blog.cloudera.com/benchmarking-nifi-performance-and-scalability/

More specifically, this possibility is enabled by the APIs and data models that the DFB adopts, which are widely accepted in the industry (e.g., Apache Kafka[15], Elastic Search API[16], compliance to Smart Data Model initiative standards[17]). Potential future implementation of the MARVEL framework may need to incorporate data streams from other third-party tools and components, such as AI components deployed outside of the MARVEL framework. In such cases, the third-party components will only need to comply with the data models specified by MARVEL and to implement corresponding Kafka Producer clients that can publish the necessary information to the DFB Kafka cluster. In addition, the DFB allows the integration of third-party front-end applications or other post-processing and visualisation tools, in cases where potential end-users with an interest in adopting MARVEL express such demands. In these cases, any third-party tool can access the real-time data managed by the DFB by connecting to the Kafka cluster via a Kafka Consumer client implementation, while historical data can be accessed by using the native Elastic Search API or the REST API of the ES proxy service that has been purpose-built for MARVEL to support a range of composite queries.

This role of the DFB adds to the overall versatility, adaptability, extensibility, scalability, and sustainability of the MARVEL framework and increases its commercialisation potential. This functionality is expected to be further analysed in T5.5.

2.2.2 Design methodology.

The general design methodology, internal architecture and constituent components of the DFB have not changed since R1 and are therefore sufficiently described in section 2.2.2 of D2.2.

2.2.3 Implementation

Work on the DFB after R1 has focused on fine tuning DFB functionalities for solving identified issues that emerged from rigorous testing activities as well as adjustments for accommodating changes in the data models of managed inference result data. In addition, data maintenance tasks have been streamlined to improve data integrity. The use of persistent volumes was introduced for storing health and performance metrics obtained from the Kafka cluster, while the foreseen retention for such data has been increased. New performance measurement features have been added which will serve the needs of benchmarking after the delivery of R2. For example, a feature has been introduced that registers the time of arrival of each message at the Kafka cluster and the time of storing the message within the Elastic Search repository in dedicated fields, which will allow the calculation of the latency introduced by the process of permanent data storage.

Furthermore, the data fusion functionality introduced in R1 has been revised and extended. This functionality refers to merging events of the same nature that are very close in time in order to form meaningful events of lower granularity that can be more effectively monitored and comprehended by human operators. This functionality has been extended to support inference results from SED and AT in addition to AVAD and ViAD which were supported in R1. Additionally, the fusion rules were revised to accommodate changes in the data models of supported AI inference results and to improve efficiency. Furthermore, elimination rules have been introduced to filter out incoming events that are considered outliers. Fused events from AVAD and ViAD are now published on Kafka topics apart from being permanently stored in Elastic Search so that they can be immediately consumed by SmartViz.

[15] https://kafka.apache.org/

[16] https://www.elastic.co/guide/en/elasticsearch/reference/current/rest-apis.html

[17] https://smartdatamodels.org/

Finally, the DFB has now been configured to support incoming data that originate from even more MARVEL AI components, leading to the support of 15 distinct data models using corresponding Kafka topics and Elastic Search indices.

In the context of R2, the DFB was successfully implemented and tested in 10 use cases in total, including 5 new use cases that were introduced after R1. Throughout R2 integration and validation tests, the DFB was found to demonstrate notable robustness, stability, and efficiency.

2.2.4 Connection to other components

The types of interaction of the DFB with other components have not changed from R1 to R2. Therefore, the description in section 2.2.4 of D2.2 is still applicable.

However, additional Kafka topics and Elastic Search indices have been added to account for inference result data originated by AI components introduced in R2. This affects the connection of the DFB to DatAna, which acts as a data producer towards the DFB, and to SmartViz and Data Corpus, that act as data consumers.

Furthermore, the ES Proxy service of the DFB was updated to address additional needs of SmartViz regarding queries of data from the DFB Elastic Search data repository.

2.2.5 Demonstration

The following figures illustrate indicative operational aspects of the DFB during runtime. In particular, Figure 7 demonstrates an indicative screenshot of the DFB monitoring UI dashboard that is based on the Grafana[18] framework and is used for monitoring the health status of the DFB Kafka cluster.

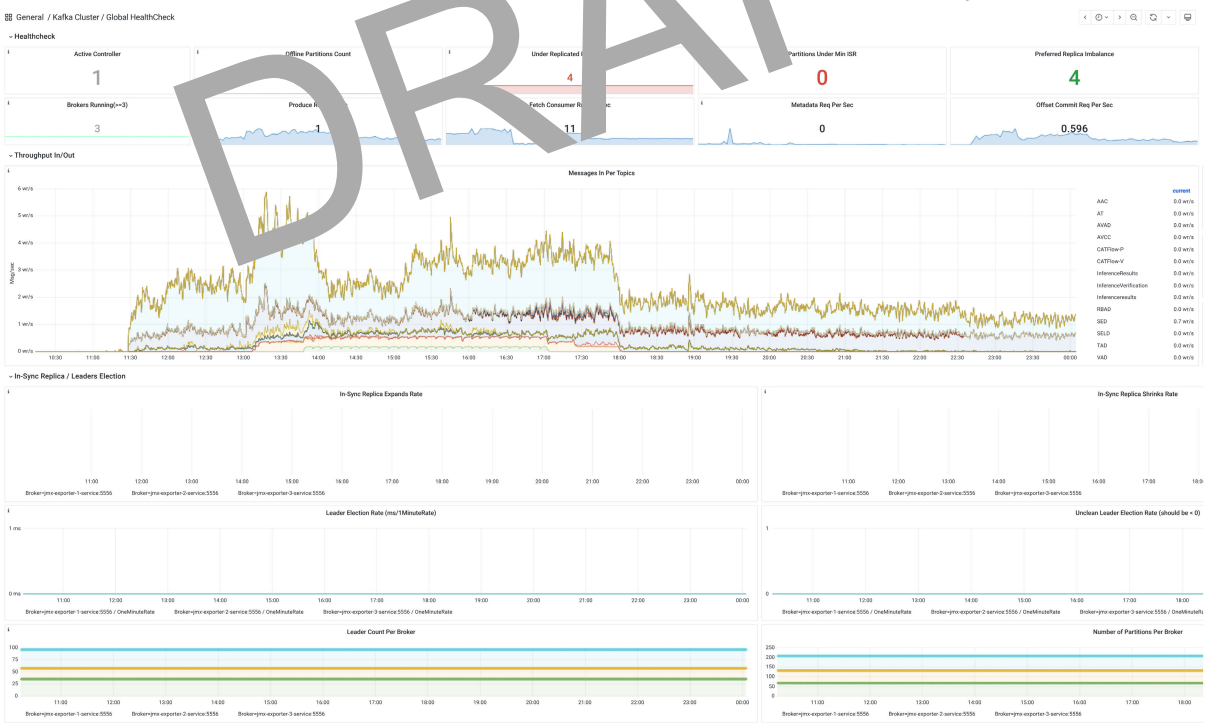


Figure 7. Indicative screenshot of the DFB health monitoring UI dashboard

[18] https://grafana.com/

Figure 8 displays a dashboard created using the DFB Grafana tool for monitoring the performance of the DFB Kafka cluster.

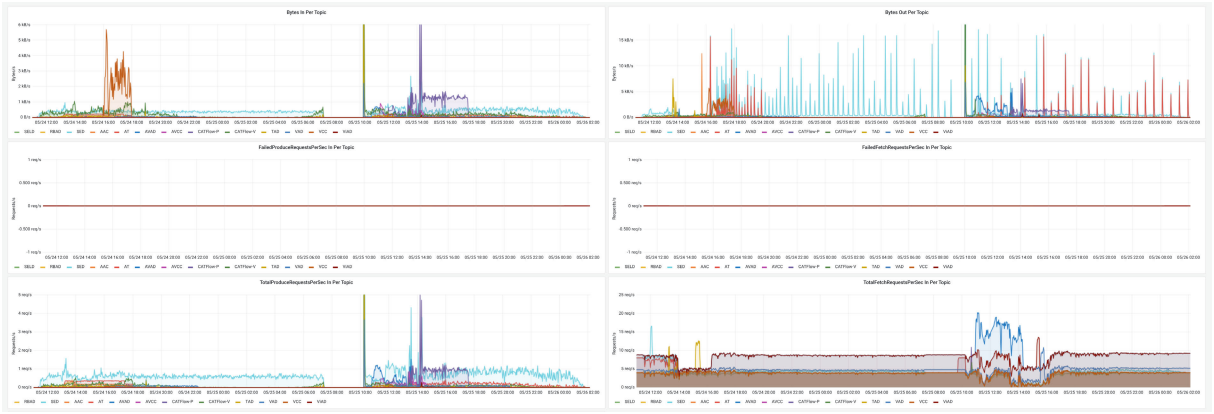


Figure 8. Kafka Cluster performance monitoring dashboard for each supported Kafka Topic over a period of 1 month. Top-Left: Bytes In. Top-Right: Bytes Out. Mid-Left: Failed Kafka Producer requests per second. Mid-Right: Failed Kafka Consumer requests per second. Bottom-Left: Total Producer requests per second. Bottom-Right: Total Consumer requests per second

Figure 9 displays a time series graph created using the DFB Grafana tool that is used for monitoring the activity of the DFB Kafka cluster and visualises the number of messages that is published on each Kafka topic.

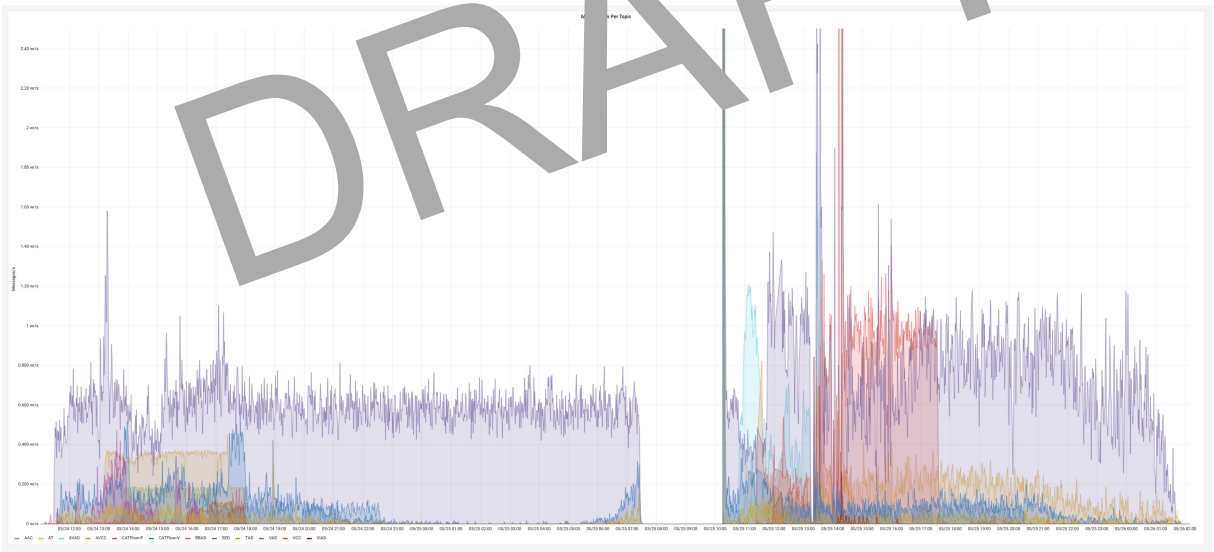


Figure 9. Number of messages published on each Kafka topic over a period of 1 month

Figure 10, Figure 11 and Figure 12 demonstrate the use of the DFB Kibana tool for visualising the distribution of generated inference results across time and for creating customised tabular views of such results. In these cases, the results originate from AT, SED and CATFlow, respectively.

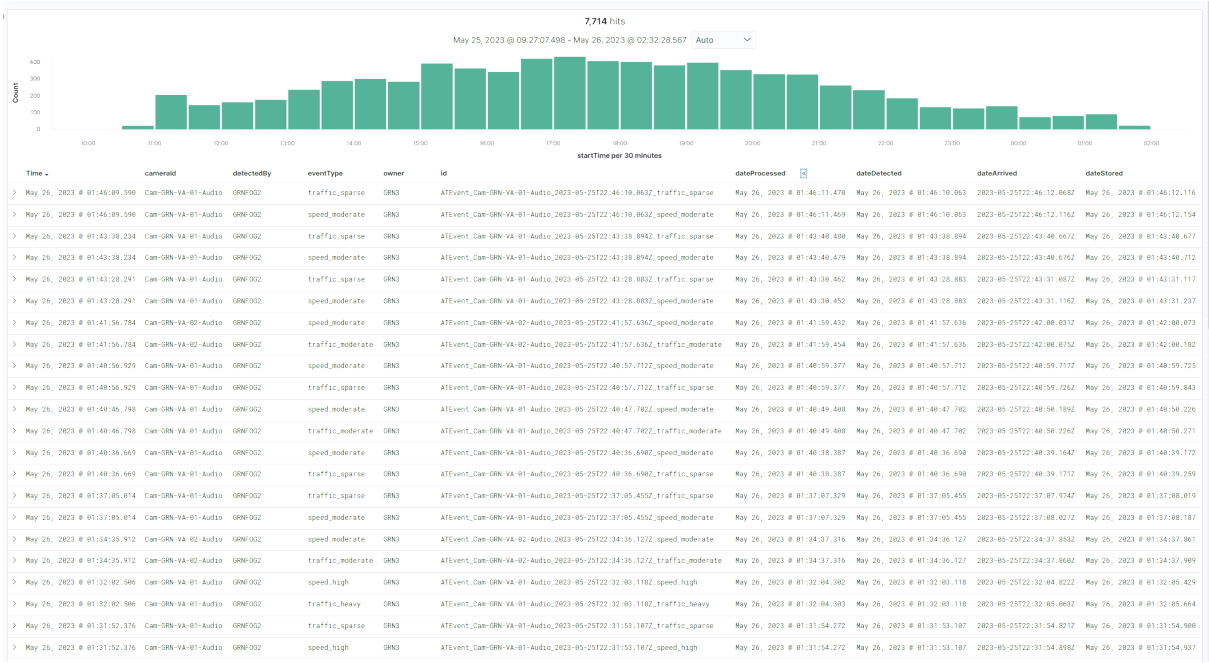


Figure 10. Inspection of inference results from AT in the DFB Kibana tool. Top: Distribution of produced events over a period of 2 days. Bottom: Configured tabular view of most recent events in given time range

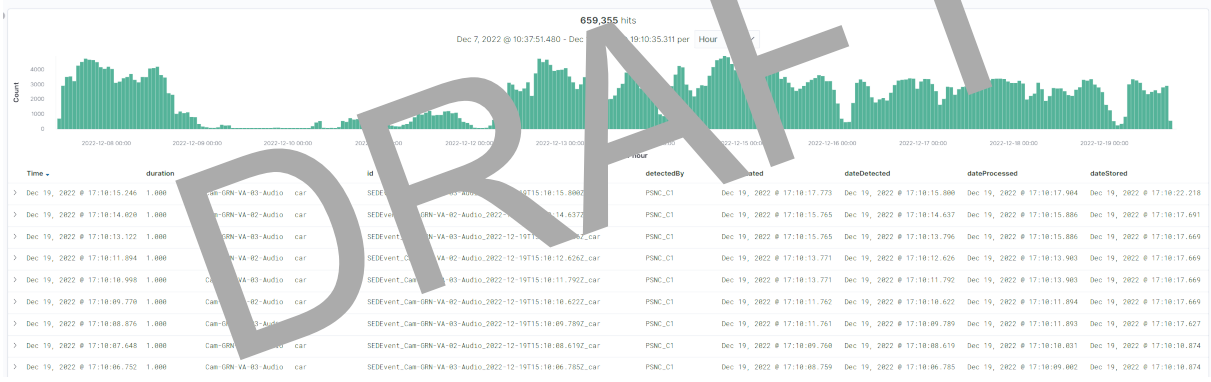


Figure 11. Inspection of inference results from SED in the DFB Kibana tool. Top: Distribution of produced events over a period of 12 days. Bottom: Configured tabular view of most recent events in given time range

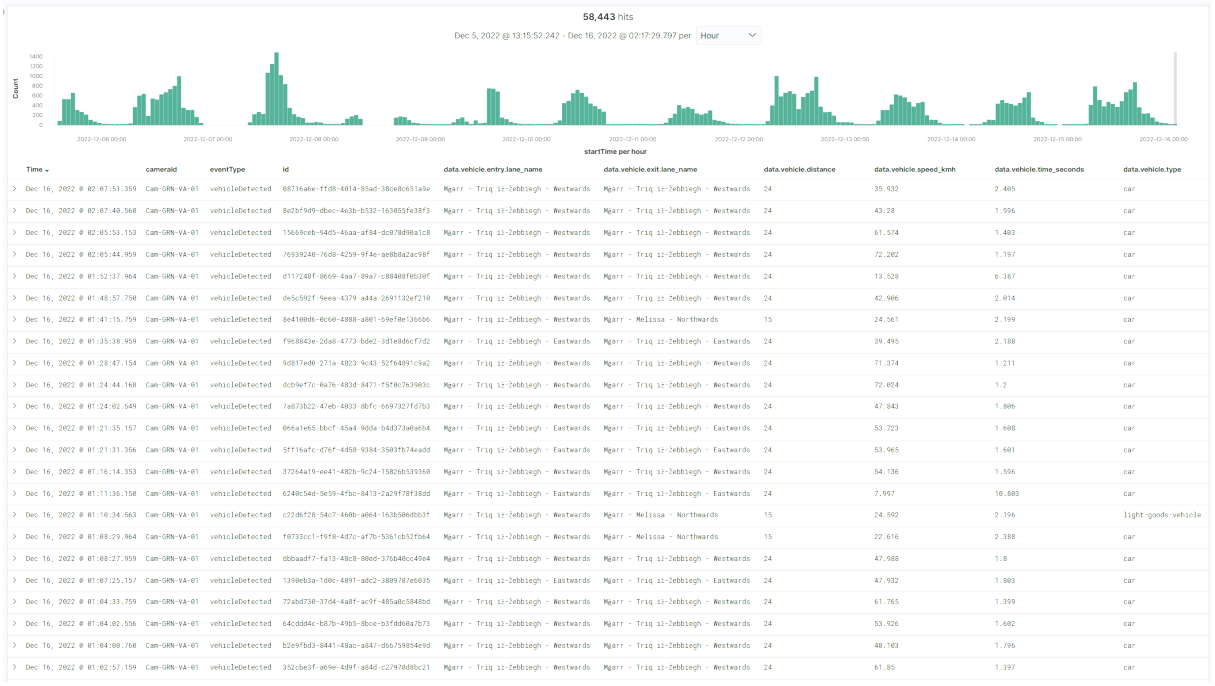


Figure 12. Inspection of inference results from CATFlow in the DFB Kibana tool. Top: Distribution of produced events over a period of 11 days. Bottom: Configured tabular view of most recent [illegible] in given time range

Figure 13 displays a dashboard created using the DFB Kibana tool for data visualisation that is connected to the Elastic Search Repository. The dashboard includes data visualisations of the number of vehicle and pedestrian detections over a period of 2 days at each of the three monitored locations of the GRN pilot, which are provided by the CATFlow component.

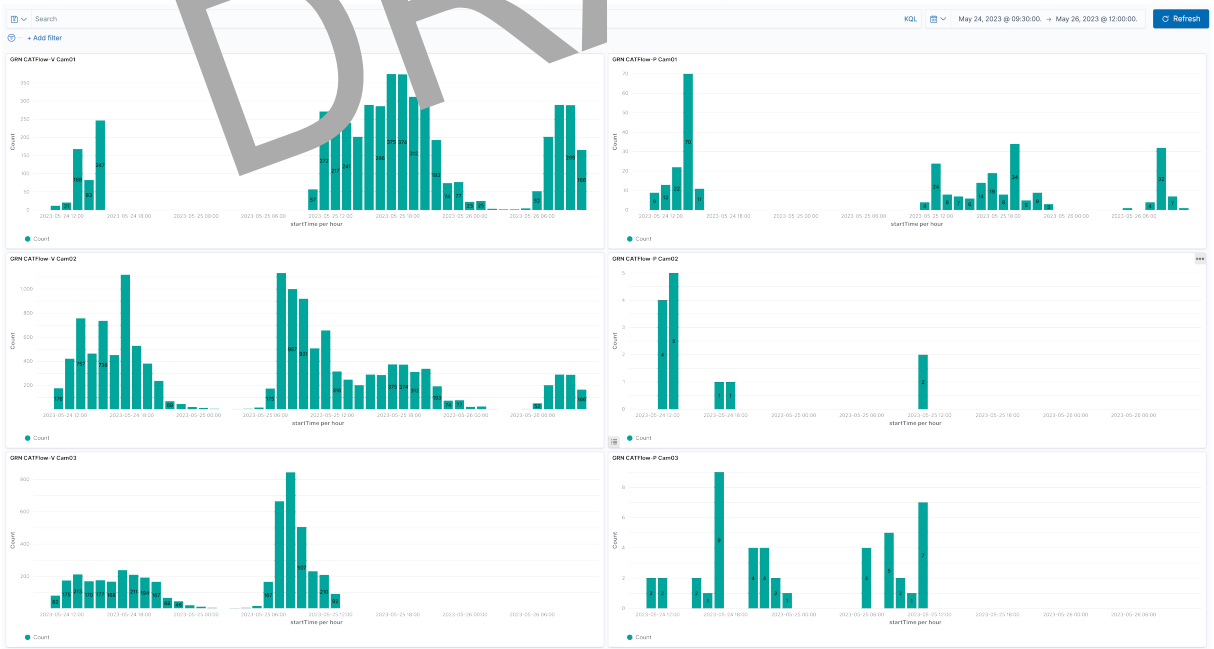


Figure 13. Dashboard created using the DFB Kibana tool displaying the number of detections of vehicles (left) and pedestrians (right) per hour by CATFlow at each of the 3 locations of the GRN pilot (Cam01, Cam02, Cam03) for a period of two days

Figure 14 illustrates a visualisation of the distribution of the number of type event types that were detected by the AT component, produced with the DFB Kibana tool.

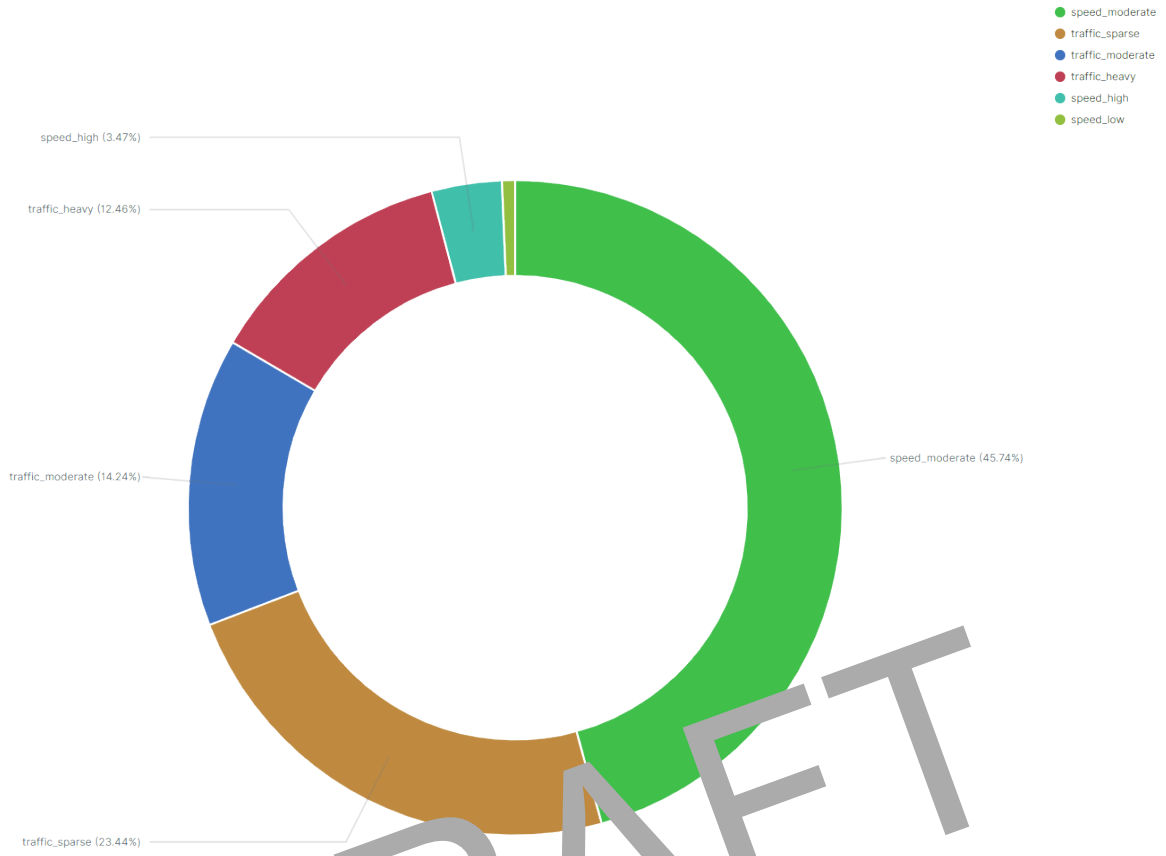


Figure 14. Data visualisation created using the DFB Kibana tool displaying the percentage of each type of event detected by the AT component over a period of 20 days in the GRN pilot

Figure 15 demonstrates the ability of the DFB Kibana tool to inspect singular events in their raw JSON format

Figure 15. Inspection of singular inference result from CATFlow in its raw JSON format within the DFB Kibana tool

2.2.6 Performance

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.

Specific, measurable metrics have been associated with the above indicators and are being illustrated in this section, along with a brief description of the purpose of each metric and the obtained measurements. These measurements were obtained from the DFB instance deployed on the cloud infrastructure provided by PSNC, within the Kubernetes cluster that was configured for MARVEL for the needs of R2 and supported by the MARVdash tool.

(a) Data Integrity KPI

Metric:

Data loss rate

Description:

Confirm that advanced encryption mechanisms over end-to-end data transfer will guarantee data integrity.

Measurement:

No data loss detected.

(b) Scalability KPI

Metrics:

HW speed up.

Description:

Increase the number of modality data streams and verify that performance metrics improve or at least stay the same.

Measurement:

The DFB in R2 supported 15 different modalities of data streams, corresponding to the available distinct streams/topics for inference results of AI components:

CATFlow-V, CATFlow-I, TAD, ViAD, AVAD, VCC, AVCC, SED, AT, VAD, RBAD, YOLO-SED, AAC, SED, GPURegex

Hardware speedup measurements are currently in progress in accordance with D5.5 guidelines. The plan is to measure latency and throughput under different load scenarios.

(c) Availability

Metrics:

Service availability-failed request, data access restriction

Description:

Verify that DFB resources are available and discoverable.

Measurement:

100% Availability

No access restriction detected.

(d) Performance for high volume, heterogeneous data streams KPI

Metrics:

Data transfer latency, data throughput, response time, number of cluster nodes.

Description:

Thoroughly measure different performance metrics under different execution conditions.

Measurement:

For this KPI and metrics, multiple measurements were taken and are analysed below.

Kafka cluster performance measurement

The Kafka cluster performance was measured with scripts developed by Gerd Koenig that are available at the following GitHub repository[19].

Kafka cluster configuration specifications

- A cluster of 3 Kafka broker nodes was used
- Kafka Topic Replication factor: 3
- Kafka Topic Partitions: 5

Kafka Producer benchmark

- **Benchmark configuration specifications**
  - Number of messages sent: 20 000
  - Message size 1024 bytes
  - acks =1
  - compression.type = lz4
  - batch.size = 100000
  - linger.ms = 50

Benchmark results:

- Throughput: 71098.471383 records/sec (69.43 MB/sec)
- Average Latency: 625.79 ms
- Max Latency: 1307.00 ms
- 50th percentile Latency: 443 ms
- 95th percentile Latency: 1292 ms
- 99th percentile Latency: 1301 ms
- 99.9th percentile Latency: 1306 ms

[19] https://github.com/gkoenig/kafka-benchmarking

Kafka Consumer benchmark

- **Benchmark configuration specifications:**
  - Messages: 10000
  - fetch-size: 1048576
- **Benchmark results:**
  - duration: 4.236 sec
  - data.consumed.in.MB: 9.8096
  - MB.sec: 2.3158
  - data.consumed.in.nMsg: 10045
  - nMsg.sec: 2371.3409"

Elastic Search performance measurement

The DFB Elastic Search Repo performance was measured using the Rally tool[20].

Benchmark configuration specifications:

- Number of docs: 7k
- doc size: 0.5 kb
- message type: http log

Throughput measurement:

- Minimum: 22.5 docs/s
- Maximum: 791.85 docs/s
- Average: 358 docs/s

Latency measurements:

- 50th percentile: 4062ms
- 95th percentile: 10534 ms
- 99.9th percentile: 13485 ms
- Average: 5193 ms.

[20] https://esrally.readthedocs.io/en/stable/

2.3 StreamHandler

INTRA's StreamHandler Platform is a complex system equipped with all the required functionalities of a state-of-the-art big data platform focusing on scalability. The main characteristics of the StreamHandler are described in the section 2.3 in D2.2, while in this deliverable the main updates since D2.2 will be demonstrated. These are a) the breakdown of the monolith architecture into microservices and b) the interconnection with DFB.

The Microservices architecture design was selected for the core functionalities of the StreamHandler deployed in the Marvel project to further enhance the robustness and efficiency of the previously deployed platform as described in D2.2.

As described in section 2.3 in D2.2, the StreamHandler Platform was designed to interoperate with many state-of-the-art IoT technologies and protocols. In this context, the interoperation of StreamHandler with the DFB took place, leading to a more robust and efficient system. Further details about the interoperation of those two components can be found in a latter subsection of this deliverable.

The division of the monolith of the core functionality to a microservice approach of the StreamHandler deployed in the MARVEL project alongside with the new interconnection with the DFB component show that the StreamHandler is a scalable and continuous evolving platform that can facilitate not only the current needs of a high-demand system but also the future needs.

2.3.1 Progress beyond the state-of-the-art

The progress beyond the state-of-the-art of the StreamHandler Platform has been thoroughly described in section 2.3.1 in D2.2. The in-depth analysis done in the context of the MARVEL framework requirements unveiled a gap that could not be covered by the DFB and DatAna components regarding the AV data management. This gap was filled by the StreamHandler being located in the Fog layer and delivering the required AV data to the cloud components. After the realisation of D2.2 a new need was discovered. This was the requirement of having precompiled AV data for all the events that occurred. The StreamHandler's interoperation with the DFB component fills the newly identified gap.

2.3.2 Design methodology.

INTRA's StreamHandler platform as described in section 2.3.2 in D2.2 was designed after thoroughly analysing all the system requirements introduced within the MARVEL framework. Based on these requirements, StreamHandler is being deployed in the fog layer of each pilot, close to the various AV data sources (AudioAnony, VideoAnony, etc.). The main tasks of the StreamHandler are to a) receive live feed through RTSP streams, segment them into smaller chunks and store them and b) make these segments available to the cloud layer components such as SmartViz and Data Corpus. In D2.2, a monolithic architecture was selected for the implementation of the StreamHandler Platform, while recently the core functions of the platform were transformed into a Microservice Architecture design, Figure 16. This makes the StreamHandler a more efficient and robust tool. Some of the benefits of this approach are:

a) Highly scalable platform. This comes from the fact that each function of the StreamHandler is a separate and independent module. In case a bottleneck is detected in a particular module, then this module can be parallelised. As an example, if we detect a lower throughput in the RTSP reception module then we can replicate this module and load balancing the streams in each submodule.
b) Fault tolerant. In case a module fails, only one of the functionalities will be affected. For example, if the SmartViz- StreamHandler module fails, then the rest of the

Platform's functionalities will remain unaffected. In addition, the microservice approach allows us to have inactive backup services in stand-by mode, so in case of a module's failure they will be initiated and the end-to-end UX will be unaffected.

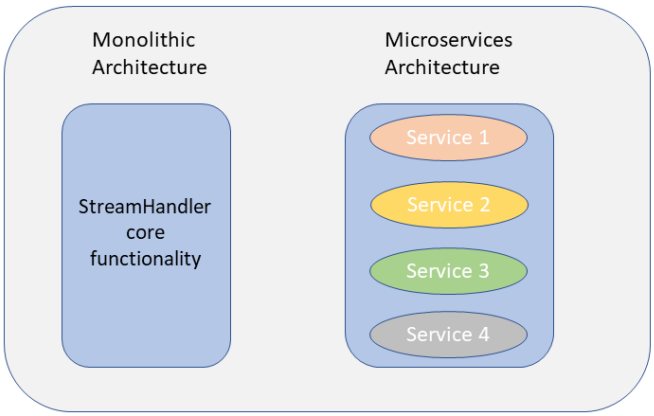


Figure 16. Monolithic vs Microservice Architectures of StreamHandler

2.3.3 Implementation

The current implementation of the modules of StreamHandler after D2.2 is totally reconstructed compared to the one presented in section 4.3.3.2 in D2.4. As described in section 2.3.2 of this deliverable, the new version is based on microservice architecture providing more flexibility and stability over the monolithic approach described in D2.2. The basic components of this newer version are:

- Java[21]
- SpringBoot[22]
- MinIO[23]
- Docker[24]
- REST API[25]
- Kafka[26]

[21] https://dev.java/

[22] https://spring.io/

[23] https://min.io/

[24] https://www.docker.com/

[25] https://restfulapi.net/

[26] https://kafka.apache.org/

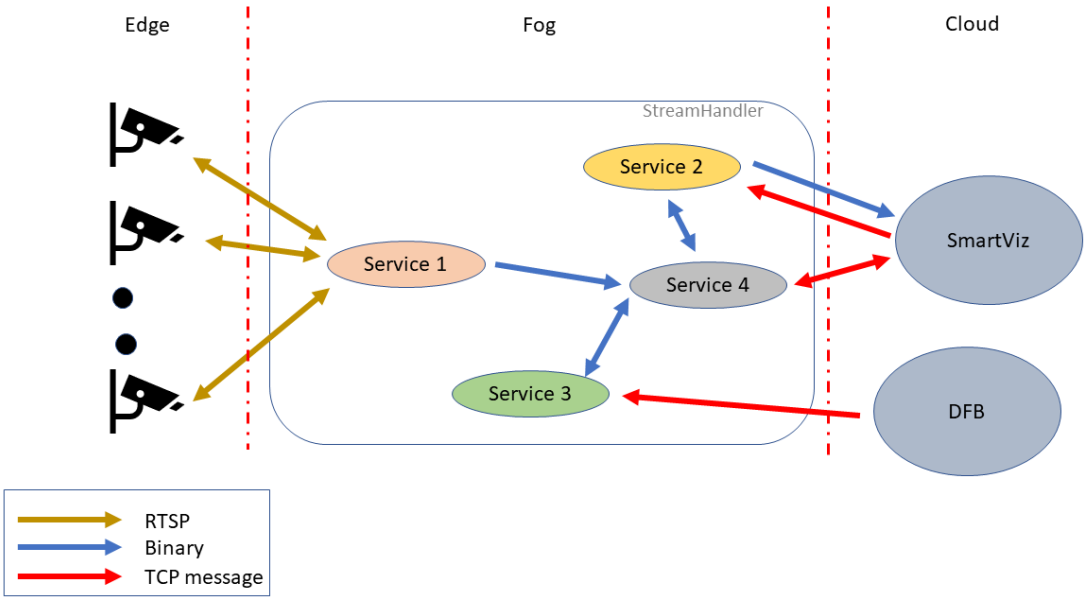


Figure 17. StreamHandler R2 implementation

As illustrated in Figure 17, the main tasks described in section 3.3.3 in D2.2 remain the same. In addition to them, one more task is added after the D2 to allow communication between the DFB component and StreamHandler. The main difference between the versions presented in D2.2 and in D2.4 is the architecture used that decouples the relation between the several services, with the only point of interaction of each service being the storage component. In detail:

- Service 1 is responsible for retrieving the IDs of the AV components from AVRegistry and initiating the segmentation procedure of each stream.

- Service 2 is responsible for providing the requested video to SmartViz through a REST API call. The service provides three REST endpoints, a) an on-demand AV creation, b) on-event AV creation and c) a list of segmented files. In the first case, the AV segments from the storage component are retrieved and a new AV file is produced at that time. The input parameters of this endpoint are: (i) sourceID (ii) timestamFrom, and (iii) timestampTo. In the second case, the endpoint receives the event ID and retrieves from the storage component the video produced by Service 3. The input parameter of this endpoint is: media_event_id. The response of both endpoints is the link of the produced file in the storage component. The last endpoint returns a JSON formatted response of the segmented file names for a specific AV source. The input parameter of this endpoint is: sourceID.

- Service 3 is responsible for constructing AV files based on events. This is achieved by listening to the Kafka messages sent when an event occurs. Then, the service based on the values shared (cameraId, startTime, endTime and timestamp if startTime, endTime are not present) constructs a new AV file.

- Service 4 is the storage component. This service is responsible for storing the AV files produced by Service 1 and Service 3. In addition, the SmartViz component is directly

retrieving from the storage component the AV file through a shared link from Service 2.

2.3.4 Connection to other components

The StreamHandler maintains the interconnections described in section 2.3.4 in D2.2 and enhanced its interconnections since D2.2. The new interconnections with the StreamHandler are:

- SmartViz: In addition to the REST API endpoint described in section 2.3.4 in D2.2, a new endpoint was added. This endpoint receives an event ID as an input parameter and responds to the SmartViz with the URL of the file in the StreamHandler's storage component.
- DFB: StreamHandler is listening for the Kafka messages advertised by the DFB component. Upon reception of such a message the StreamHandler is producing a new AV file based on the camera ID, start/end time parameters found in the Kafka message.

2.3.5 Demonstration

INTRA's StreamHandler has been deployed in all Pilots, i.e., GRN, MT, and UNS. As described in subsection 2.3.2, the StreamHandler has been deployed as microservices. In Listing 1 and Listing 2 the YAML files of the main microservices are presented. In Figure 18, Figure 19 and Figure 20 a description of the developed APIs is provid[illegible]. The documentation was created using the SWAGGER[27] API tool considering thr[illegible]se c[illegible]es. In Figure 21, the directory orchestration is illustrated. As can be seen, t[illegible] directory seg[illegible]ents contain all the videos streamed from the AI components segmented i[illegible] predefined du[illegible]tion. The directory "generated-media" contains the AV files creat[illegible]sed on [illegible]artViz's requ[illegible]ts and the AV files created based on the event det[illegible]ech[illegible]ms [illegible]f the [illegible] components developed under the MARVEL framework. In Fi[illegible]re 22, a [illegible]mple [illegible]f t[illegible]s stored is presented.


```
# media-conte[illegible]t-servic[illegible]-gr[illegible] yaml
apiVersion: v[illegible]
kind: Service
metadata:
  name: $NAME
spec:
  type: ClusterIP
  selector:
    app: $NAME
  ports:
    - name: http
      protocol: TCP
      port: 8889
      targetPort: 8080
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: $NAME
```

[27] https://swagger.io/

```
spec:
  replicas: 1
  selector:
    matchLabels:
      app: $NAME
  template:
    metadata:
      labels:
        app: $NAME
    spec:
      containers:
      - name: $NAME
        image: 192.168.50.1:5000/media-content-service:1.0
        volumeMounts:

            - name: ${PRIVATE_VOLUME}
              mountPath: /conf.json
              subPath: conf/conf.json
        ports:
        - containerPort: 8080
      affinity:
        nodeAffinity:
         requiredDuringSchedulingIgnoredDuringExecution:
          nodeSelectorTerms:
          - matchExpressions:
            - key: Layer
              operator: In
              values:
              - GRNFOG
      tolerations:
      - key: "Layer"
        operator: "Equal"
        value: "GRNFOG2"
```

```
kind: Template
name: media-content-services-grn
description: media-content-service-grn
variables:
- name: NAME
  default: media-content-service-grn
- name: PRIVATE_VOLUME
  default: private-volume
```

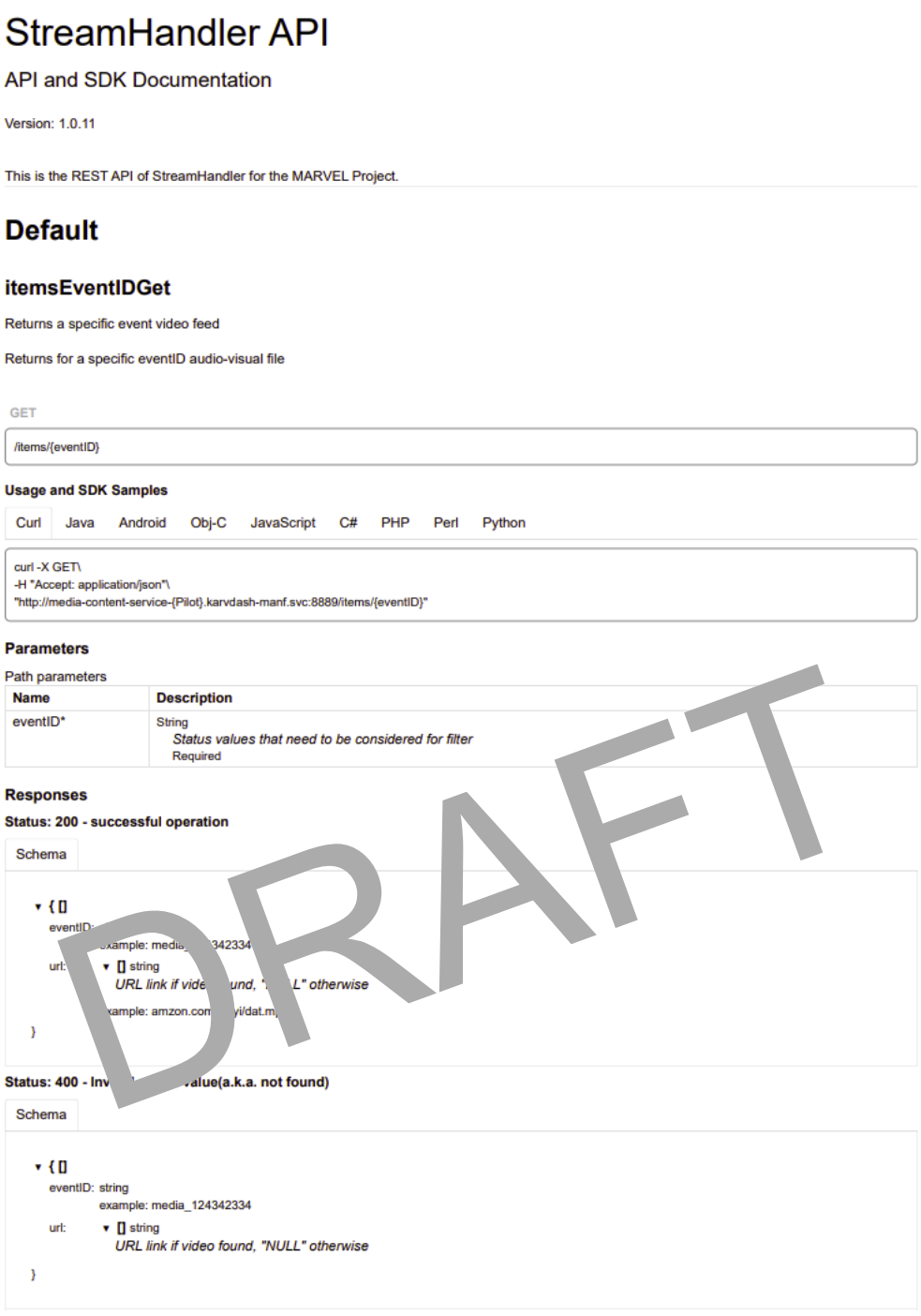
Listing 1. StreaamHandler's API service example YAML file

```
# rtsp-stream-handler-grn.yaml
apiVersion: v1
kind: Service
metadata:
  name: $NAME
spec:
  type: ClusterIP
  selector:
    app: $NAME
  ports:
    - name: http
      protocol: TCP
      port: 8881
      targetPort: 8080
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: $NAME
spec:
  replicas: 1
  selector:
    matchLabels:
      app: $NAME
  template:
    metadata:
      labels:
        app: $NAME
    spec:
      containers:
      - name: $NAME
        image: 192.168.50.1:5000/rtsp-stream-handler-grn:1.0
        volumeMounts:

            - name: ${PRIVATE_VOLUME}
              mountPath: /conf.json
              subPath: conf/conf.json
        ports:
        - containerPort: 8080
      affinity:
        nodeAffinity:
         requiredDuringSchedulingIgnoredDuringExecution:
          nodeSelectorTerms:
          - matchExpressions:
            - key: Layer
              operator: In
              values:
              - GRNFOG2
      tolerations:
```

```
      - key: “Layer”
        operator: “Equal”
        value: “GRNFOG2”
---
kind: Template
name: rtsp-stream-handler-grn
description: rtsp-stream-handler-grn
variables:
- name: NAME
  default: rtsp-stream-handler-grn
- name: PRIVATE_VOLUME
  default: private-volume
```

Listing 2. StreamHandler’s Segmentation service example YAML file



StreamHandler API

API and SDK Documentation

Version: 1.0.11

This is the REST API of StreamHandler for the MARVEL Project.

Default

itemsEventIDGet

Returns a specific event video feed

Returns for a specific eventID audio-visual file

GET

/items/{eventID}

Usage and SDK Samples

Curl | Java | Android | Obj-C | JavaScript | C# | PHP | Perl | Python

```
curl -X GET\
-H "Accept: application/json"\
"http://media-content-service-{Pilot}.karvdash-manf.svc:8889/items/{eventID}"
```

Parameters

Path parameters

Name	Description
eventID*	String *Status values that need to be considered for filter* Required

Responses

Status: 200 - successful operation

Schema

```
▾ { []
  eventID:
          example: media_124342334
  url:    ▾ [] string
            URL link if video found, "NULL" otherwise
          example: amzon.com/dat.m
}
```

Status: 400 - Invalid value(a.k.a. not found)

Schema

```
▾ { []
  eventID: string
           example: media_124342334
  url:     ▾ [] string
             URL link if video found, "NULL" otherwise
}
```


Figure 18. StreamHandler's API documentation (part1)

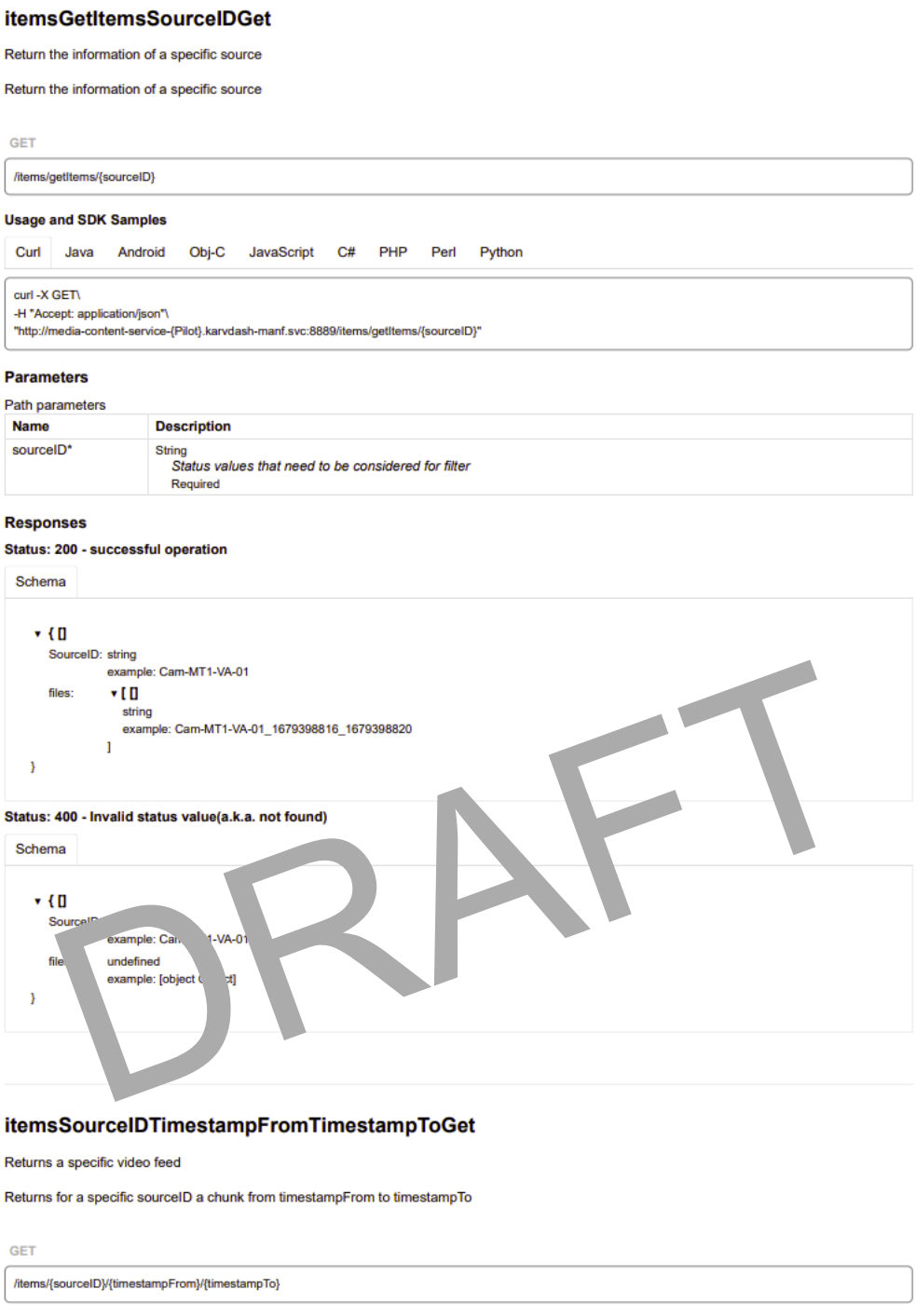


Figure 19. StreamHandler's API documentation (part 2)

Usage and SDK Samples

Curl | Java | Android | Obj-C | JavaScript | C# | PHP | Perl | Python

```
curl -X GET\
-H "Accept: application/json"\
"http://media-content-service-{Pilot}.karvdash-manf.svc:8889/items/{sourceID}/{timestampFrom}/{timestampTo}"
```

Parameters

Path parameters

Name	Description
sourceID*	String *Status values that need to be considered for filter* Required
timestampFrom*	String *Status values that need to be considered for filter* Required
timestampTo*	String *Status values that need to be considered for filter* Required

Responses

Status: 200 - successful operation

Schema

▾ {}
sourceID: string
example: Cam-MT3-MEMS-01
timestampFrom: string (date-time)
example: 2022-04-01T10:08:24.62Z
timestampTo: string (date-time)
example: 2022-04-01T10:08:34.62Z
url: ▾ [] string
URL link if video found, "NULL" otherwise
example: amzon.com/jsdyi/dat.mp4
}

Status: 400 - Invalid status value(a.k.a. not found)

Schema

▾ {}
sourceID: string
}

Figure 20. StreamHandler's API documentation (part 3)

```
drwxr-xr-x 7 root root    4096 ??? 20 08:45 .
drwxrwxrwx 7 ics  ics     4096 ??? 14 11:28 ..
drwxr-xr-x 2 root root    4096 ??? 15 12:14 Cam-UNS-AA-01
drwxr-xr-x 2 root root   12288 ??? 15 10:52 Cam-UNS-AA-02
drwxr-xr-x 2 root root 1073152 ??? 17 00:48 Cam-UNS-VA-01
drwxr-xr-x 2 root root    4096 ??? 15 14:38 generated-media
```

Figure 21. Sample of the directories created per AV source in UNS Fog Pilot

Figure 22. Sample of the segmented AV files produced by StreamHandler

2.3.6 Performance

StreamHandler is in its second version. During its development and deployment several performance tests were carried out. As an indicative sample, the performance results acquired in the examined scenario are reported in Table 2. In our scenario, a four-core CPU machine was used with AV MPEG-TSP input streams. The following four use cases were examined:

a. No load: in this scenario we examined the resources consumed by the Platform without any RTSP traffic. The results of this scenario were used as a baseline for the later results.

b. One stream: in this scenario, we considered that there is only one stream traffic load.

c. Two streams. In this scenario, we considered that there are two streams contributing as traffic load to the platform.

d. Three streams. In this scenario, we considered that the platform is under load, serving three streams in parallel.

Table 2: StreamHandler's use case testing results

Use case	CPU %	Memory %	NET I/O	Block I/O
A	3.18%	1.97%	876B	0 B
B	3.61%	2.18%	1.69MB	410KB
C	4.06%	2.18%	2.77MB	3.42MB
D	4.28%	2.19%	5MB	5.51MB

The initial tests demonstrate that StreamHandler is a robust and efficient component. Further results regarding StreamHandler's performance can be found within the WP5 deliverables.

2.4 HDD

The current version of our HDD expands the one described in D2.2 in several key ways. Firstly, we have introduced a new methodology for the automated performance evaluation of Apache Kafka clusters. This methodology allowed us to run experiments in an automatised prototype to compare BroMax[28] and BroMin with different replication factors, and numbers of consumers. By doing so, we have been able to use scalable data and cross-check our MARVEL-tailored simulation results of D2.2. Secondly, we have included the results of these experiments, which have highlighted some qualitative differences with respect to the conclusions obtained with the MARVEL-tailored simulation model alone. These differences emphasise the importance of cross-checking analysis with real-life data before production deployment. Lastly, we have made our evaluation framework[29] available to the community so that others can use it for their own performance evaluations. Overall, these additions provide a more complete and robust analysis of Apache Kafka topic partitioning and demonstrate the practical implications of our simulation-based approach.

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. For an exact novelty positioning, the reader can refer to section 2.4.1 in D2.2.

2.4.2 Design methodology.

To extend the experimenting ability of the HDD we designed an open-source, automatised, prototype implementation. The implementation consists of two high-end servers: one hosting the client-side scripts and tools and another handling the cluster-side, i.e., the Apache Kafka brokers and a Zookeeper instance for leader election among them. The software on the cluster-side server runs within Docker containers configured with Docker Compose[30]. This approach is suitable for running the entire cluster within a single physical server, but the scripts we developed can be adapted to match the specific characteristics of the target deployment under test, e.g., a distributed environment where Apache Kafka is run within a Kubernetes cluster. The methodology would remain the same and it is illustrated by means of the sequence diagram in Figure 23 (notation explained in section 2.4.2 in D2.2), which is entirely managed through the execution of a single Bash script on the client-side server:

[28] https://arxiv.org/pdf/2205.09415.pdf

[29] https://github.com/ccicconetti/kafka-hdd

[30] https://docs.docker.com/compose/

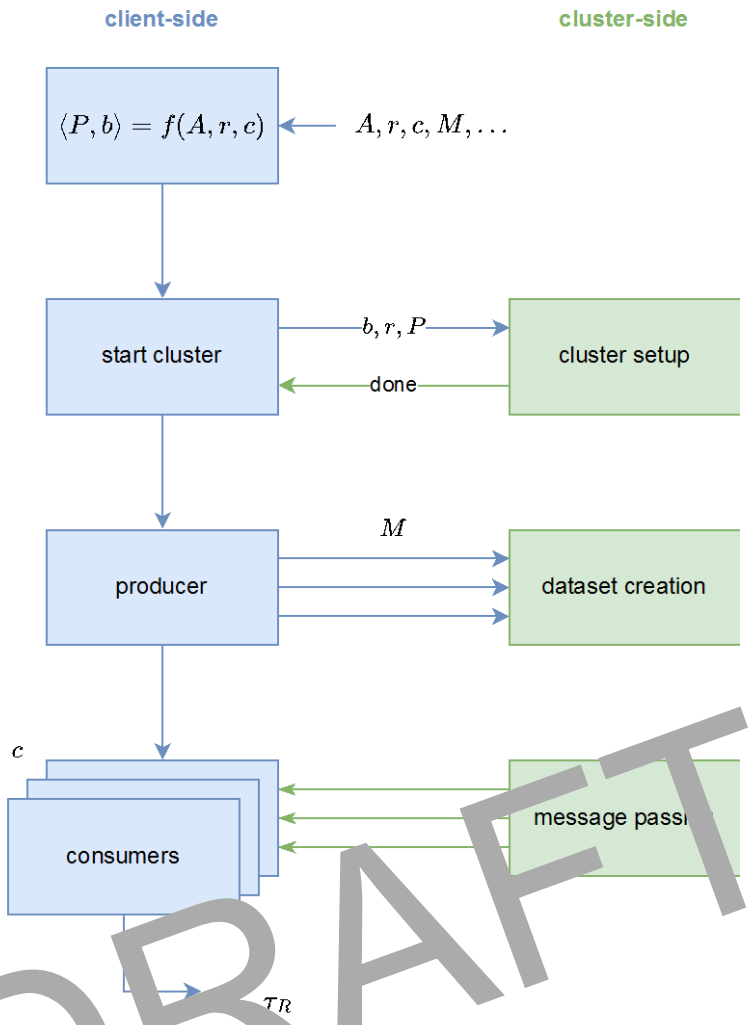


Figure 23. Methodology adopted for the execution of automated experiments with an Apache Kafka cluster

2.4.3 Implementation

At the beginning of the process, there is no Apache Kafka cluster running. In principle, there are situations where we could reuse a running cluster from the previous experiment, i.e., when the cluster parameters *P and b* remain the same, but we decided to start with clean conditions to ensure independence and repeatability. The inputs of the process are: the Apache Kafka cluster provisioning algorithm *A*, the replication factor *r*, the number of consumers *c,* and the size of the Apache Kafka messages exchanged *M*, in bytes. The algorithms also require additional system parameters, as described in D2.2. On the client, the algorithm *A*, i.e., BroMin or BroMax, is run to determine the number of topic partitions *P* and brokers *B*. The client issues commands to remotely start the Apache Kafka cluster of *b* brokers via Docker Compose[31], creates the topic that will be used in the current experiment, and configures it with the given replication factor *r* and the number of partitions *P*. Once done, the client starts a producer to fill the topic with a dataset of messages of size *M*. When the dataset is complete, the core part of the experiment starts. It is carried out by means of a pool of *c* consumers, all belonging to the same consumer group, started as independent processes with a small random initial delay. All the consumers greedily read messages from the dataset as fast as they can, without performing

[31] https://docs.docker.com/compose/

any processing on the payload or saving it. This is because the experiment is intended to measure an upper bound of the consumer throughput T_c, which is the final output together with the rebalance time. The latter is the time needed for the assignment of the partitions to the consumers to converge to its final configuration.

2.4.4 Connection to other components

As described in D2.2 and D5.4, HDD provides optimisation recommendations to DFB, as per the application settings. Specifically, HDD interacts with the DFB as follows: (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

We conducted some experiments to demonstrate the new findings. We have set the maximum number of available brokers B to 16, based on the hardware characteristics of the cluster-side server (a Supermicro server with four CPUs AMD Opteron(tm) Processor 6282 SE and 128 GB RAM) and we varied the number of consumers c in the range [25,125].

In Figure 24, we show the output of the BroMax and BroMin algorithms, i.e., P on the left-side plot and b on the right-side one, with two replication factors: r=3, which is a typical value used in production systems, and r=5, which is intended for the dispatch of critical messages in a system where brokers are prone to failures. The findings are reported in the next subsection.

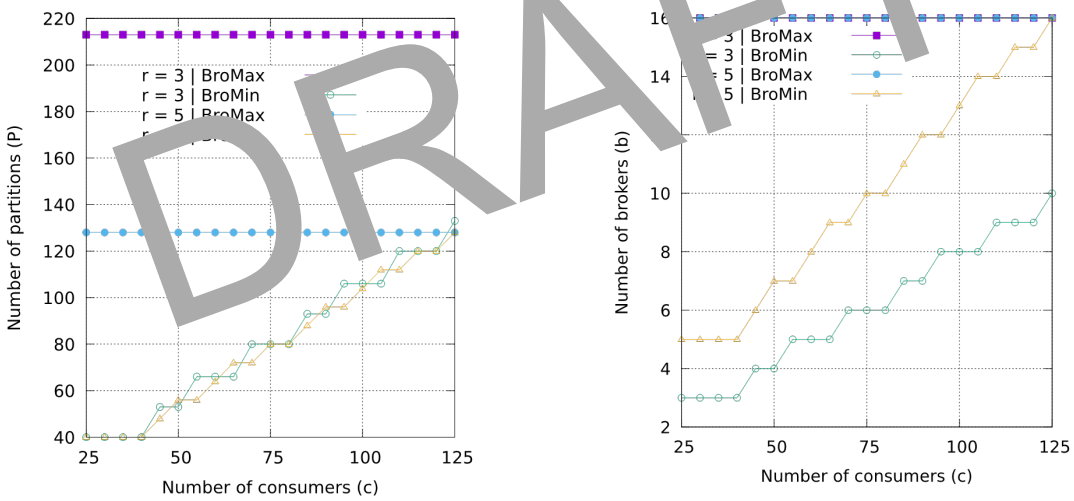


Figure 24. Number of partitions and brokers selected by the BroMin/BroMax algorithms and used in the experiments

2.4.6 Performance

We perform an analysis of results with messages of M=1 kB size. In Figure 25, we show the consumer throughput T_c. As can be seen, all the curves decrease with increasing number of consumers c. This is an expected behaviour with small message sizes and lightweight consumers because they can ingest messages very fast, thus a more aggressive parallelisation results in a negligible speed-up compared to a noticeable overhead. Furthermore, BroMax always achieves the lowest T_c, even though it always saturates the number of brokers, i.e., b=B=16 in the experiments, and uses more partitions for the topic. There are some low-level parameters in Apache Kafka that could be adjusted to fine-tune the communication

performance, such as the size of the buffers used by TCP sockets. However, to the best of our knowledge, there are no widely used guidelines that can be adopted to optimise performance metrics according to a specific scenario, thus we have left the default values and did not consider them as part of our analysis. Finally, the impact of the replication factor r=3 vs. r=5 is small with BroMax and negligible with BroMin. Again, this is because of the small size of messages, which keeps the additional overhead due to an increased replication low, especially with fewer brokers/partitions, i.e., in the BroMin case.

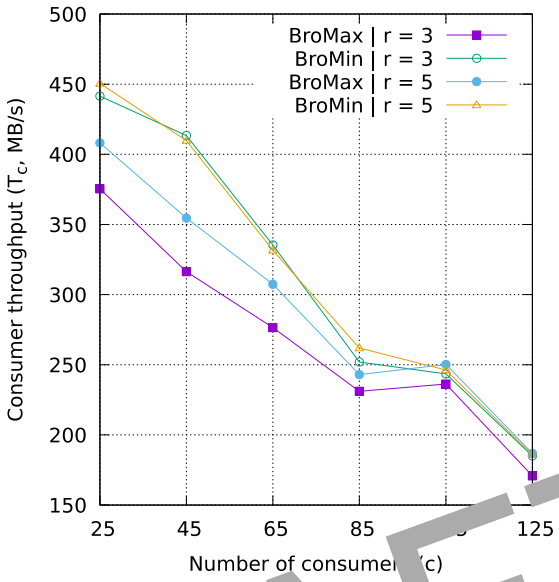


Figure 25. Co u r throug

In Figure 26, we show the r balance t ne τ_R. Ba d the esults, the replication factor does not significant ffect is m tric the ot r hand, BroMax requires a significantly longer time for the co umer gr p to onverg a able distribution of partitions than BroMin with a small numbe f consun rs, i.e., as $c < 85$. This can be explained by the higher number of brokers. Howeve as the umbe of consumers becomes very large, i.e., in the right part of the plot, the τ_R curves e almost overlapping, due to the counterbalancing effect of the number of consumers. Remember that in our experiments the consumers join the group with small random offsets and every new consumer entering the group will trigger a rebalancing procedure: more consumers result in a higher rate of rebalancing events triggered.

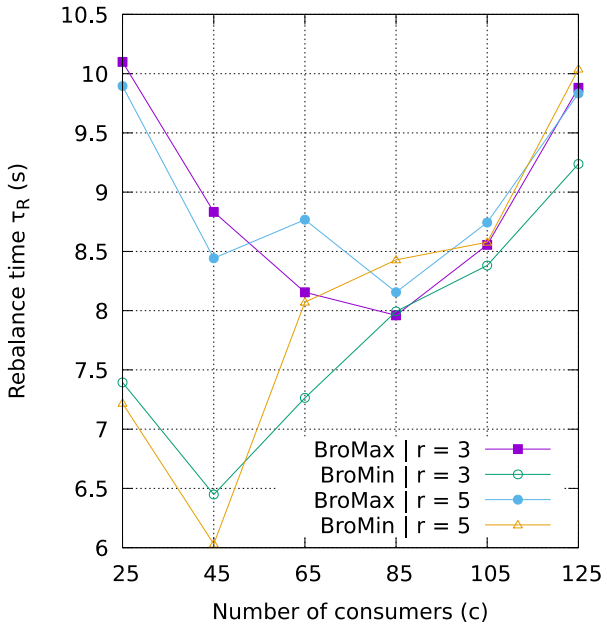


Figure 26. Rebalance time

In Figure 27, we report a representative snapshot of the CPU load measured on the physical servers, which were unloaded except for the Apache Kafka prototype experiments. The valleys correspond to when the experiment iterations terminate after which the load increases up to peaks that mark the end of the respective experiments. In addition to following, the same time pattern, incidentally, the CPU load of the client and server nodes have similar values. We included this plot to show that neither of the two servers is overloaded (the server has 64 real CPU cores), which otherwise would have affected the results.

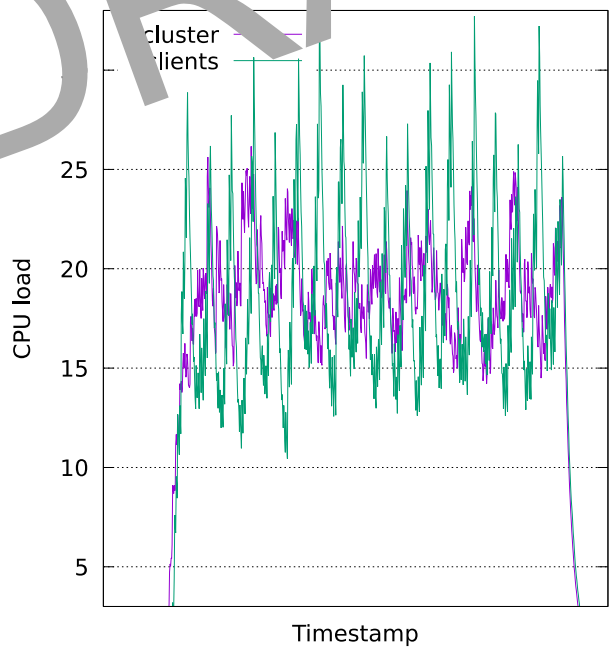


Figure 27. CPU load on the physical servers

For the same reason, we report in Figure 28 the disk I/O on the cluster server. This metric is irrelevant on the client side because the clients do not use the disk at all as they immediately

drop the payload of messages after ingesting them. Also, for this metric, we can see that the server cannot be overloaded, and in fact, Apache Kafka is known to be quite efficient in this respect despite relying heavily on data persistence on disk.

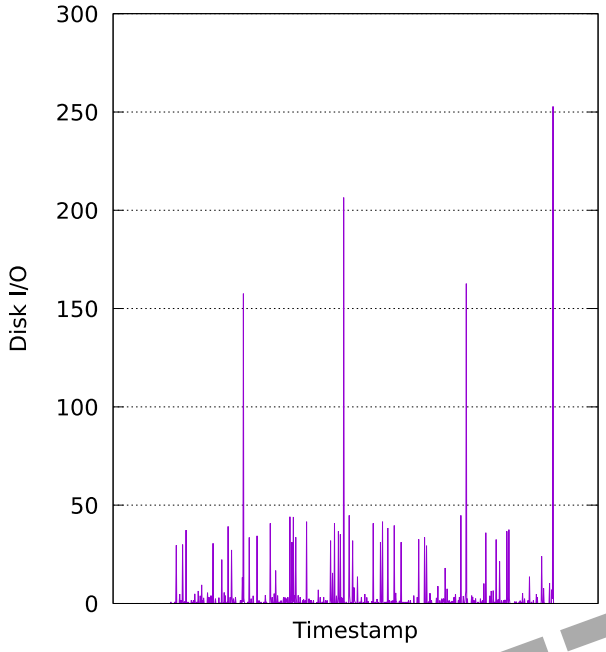


Figure 28. Disk I/O load on the broker server

3 Positioning within the E2F2C continuum

3.1 The DMP toolkit architectural approach

The architectural approach followed for the application of the DMP in the R1 inference pipeline was found to be effective and has remained intact in R2, being applied in all 10 R2 use cases. Therefore, the description in section 4.1 of D2.2 accurately describes the current DMP internal architecture and overall operation.

The DMP has been applied to 5 new use cases introduced for R2, and this has led to the support of additional AI components and respective data models of their raw inference results. Nevertheless, the same data models from the Smart Data Models (SDM) initiative, supported by the FIWARE foundation[32] that had been applied in R1 (Alert, Anomaly, MediaEvent data models) could cover the needs of the added AI components. Therefore, DatAna processes the raw inference results of the new AI components to transform them into the 3 SDM-compliant data model counterparts before relaying them to the DFB.

One notable change concerns StreamHandler and its revised internal operation. The updated version of StreamHandler now interacts with the DFB by subscribing to the Kafka topics where AI inference results are published in order to consume these results and generate AV files that correspond to these results. Therefore, StreamHandler acts proactively and AV files are no longer created only at the request of SmartViz. Instead, in R2, SmartViz can also issue a request to StreamHandler for an AV file by submitting the id of an inference result, for which the AV file is required. This change has been reflected in the revised REST communication protocol between StreamHandler and SmartViz.

The DMP has been applied in 10 use cases derived for the needs of the final version of the MARVEL Integrated framework (R2). The following figures illustrate the system architecture diagrams that were specified to address these 10 use cases, which include the specific configuration of the DMP toolkit for each case along with the associated DMP components (depicted in blue colour).

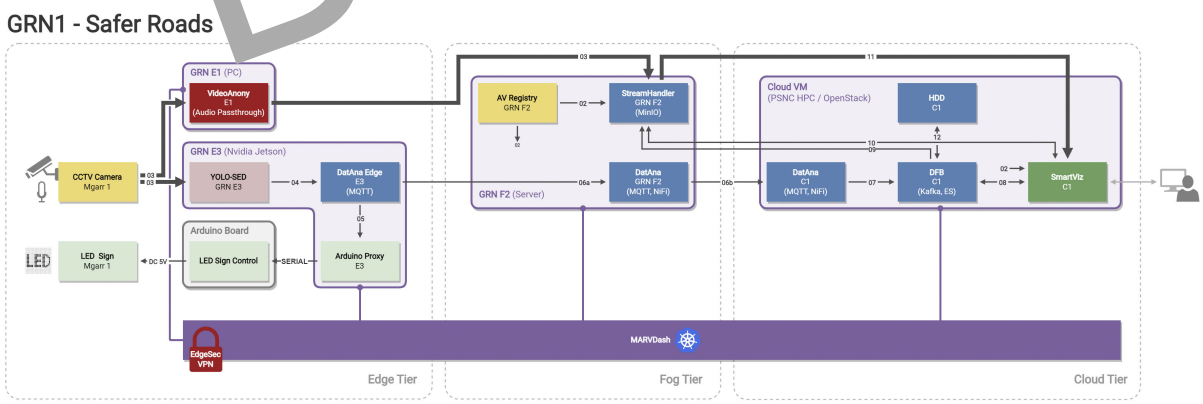


Figure 29. The DMP in the overall system architecture for use case GRN1: Safer Roads

[32] https://www.fiware.org/

GRN2 - Road user behaviour

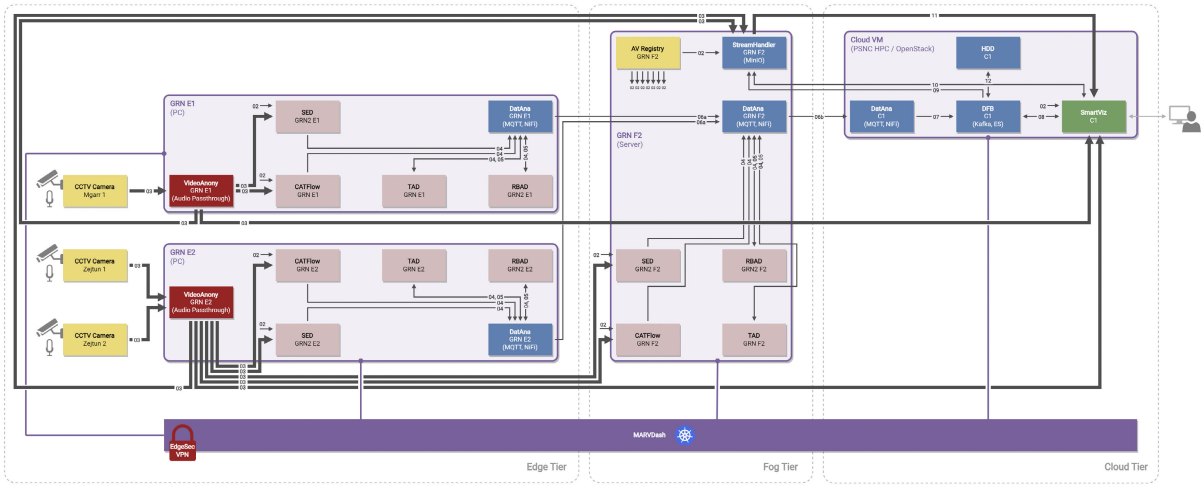


Figure 30. The DMP in the overall system architecture for use case GRN2: Road user behaviour

GRN3 - Traffic Conditions and Anomalous Events

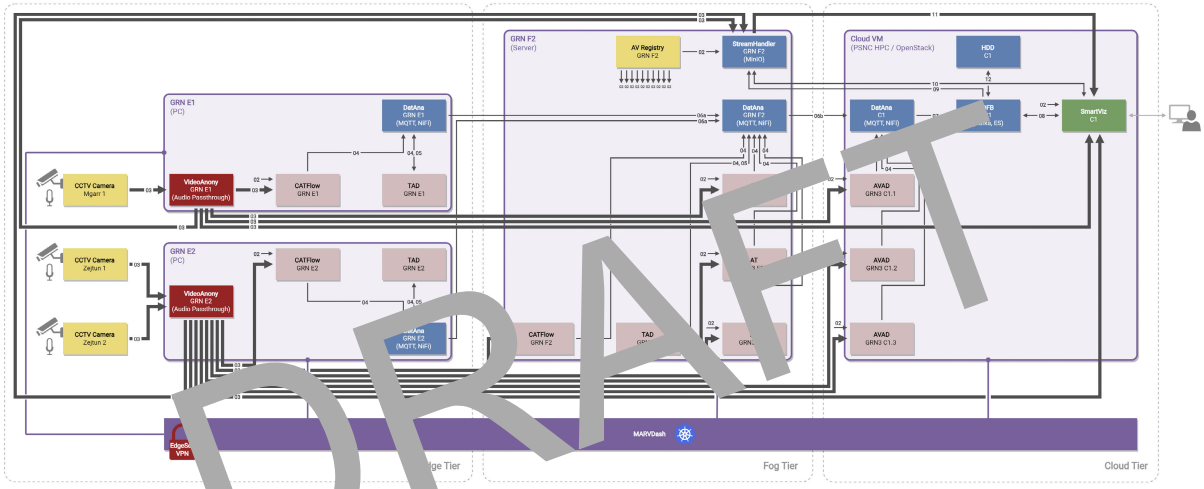


Figure 31. The DM in the erall system architecture for use case GRN3: Traffic Conditions and Anomalous Events

GRN 4 - Junction Traffic Trajectory Collection

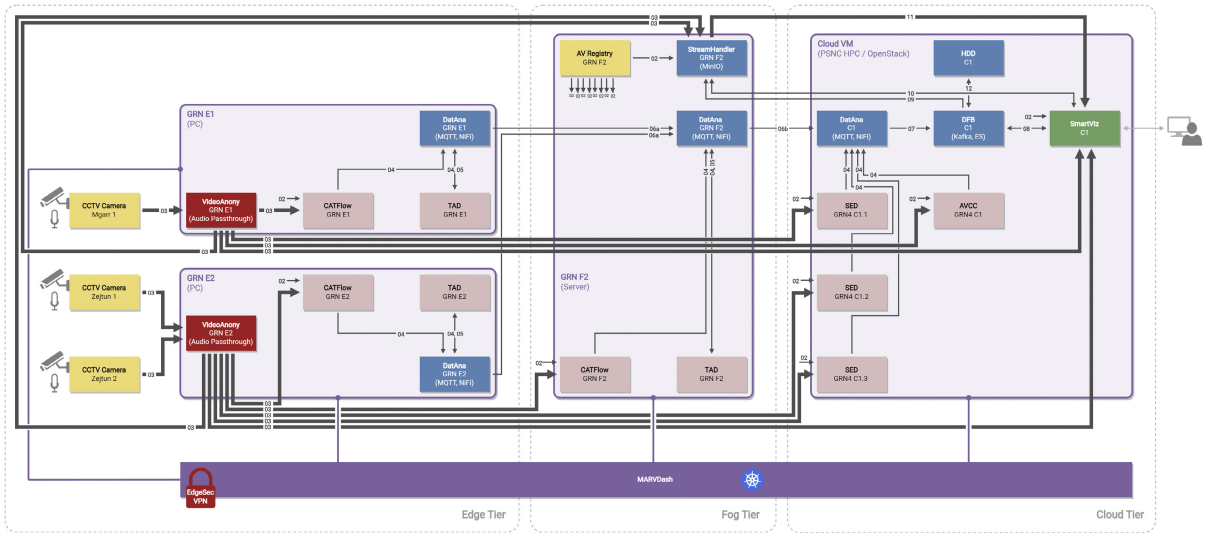


Figure 32. The DMP in the overall system architecture for use case GRN4: Junction Traffic Trajectory Collection

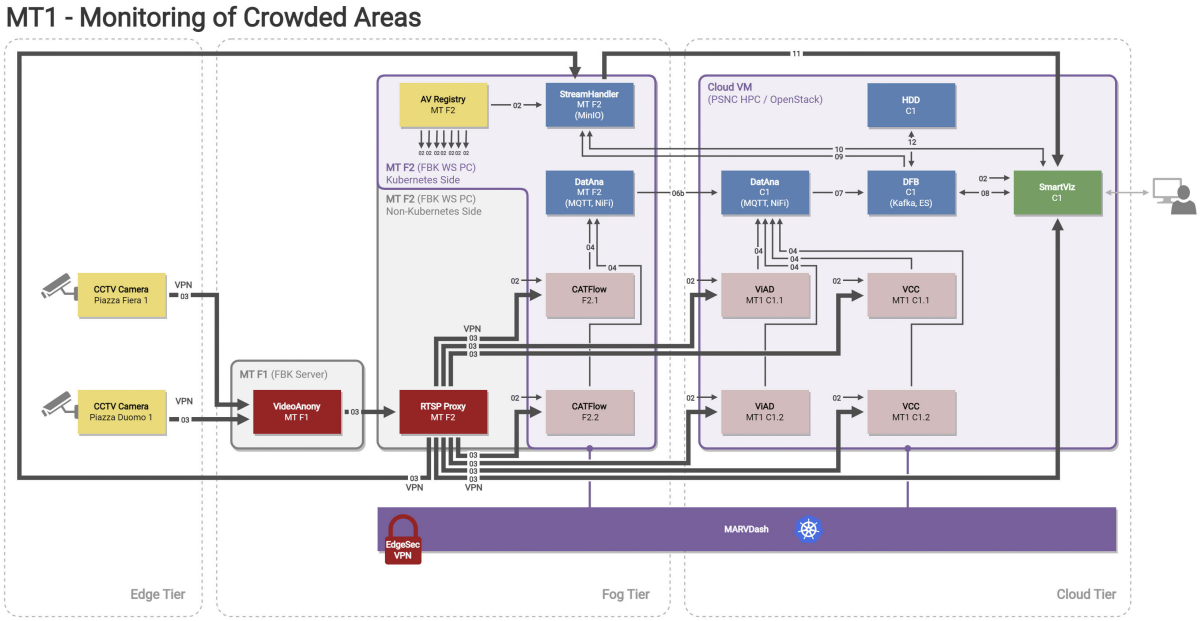


Figure 33. The DMP in the overall system architecture for use case MT1: Monitoring of crowded areas

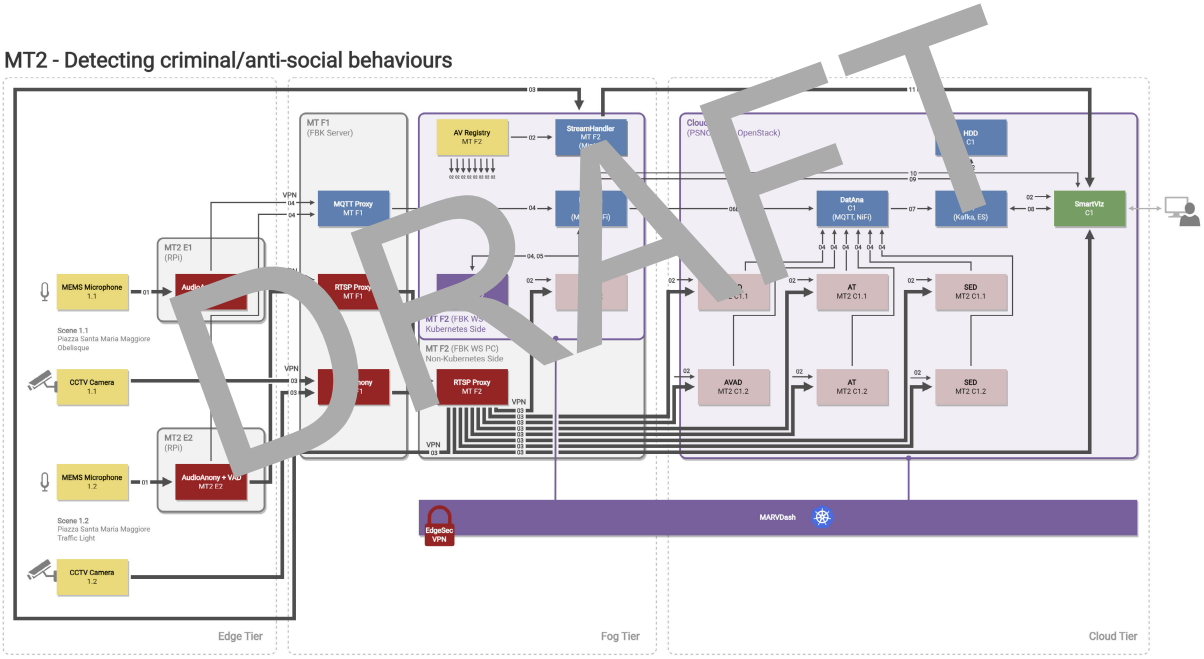


Figure 34. The DMP in the overall system architecture for use case MT2: Detecting criminal/anti-social behaviours

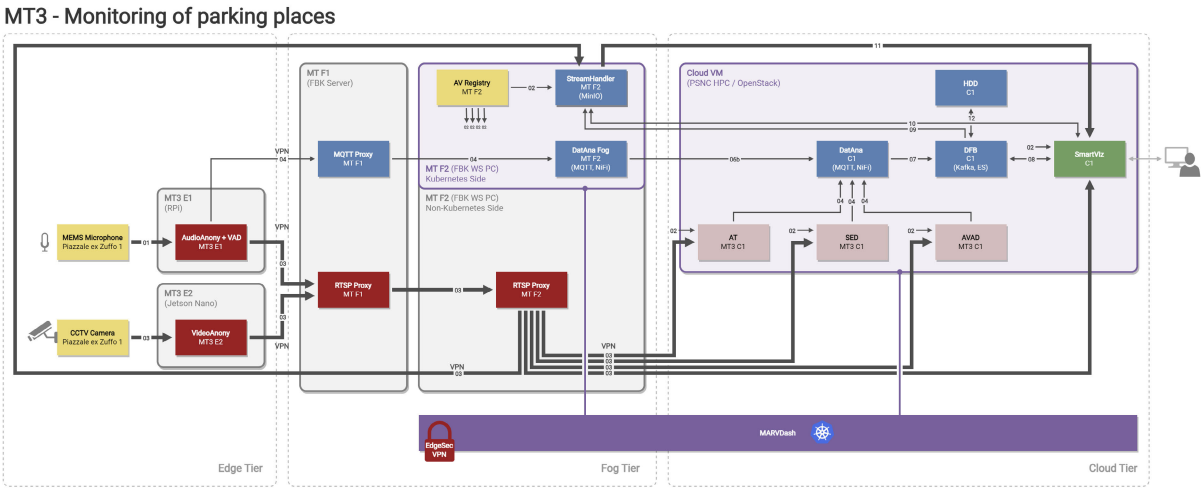


Figure 35. The DMP in the overall system architecture for use case MT3: Monitoring of parking places

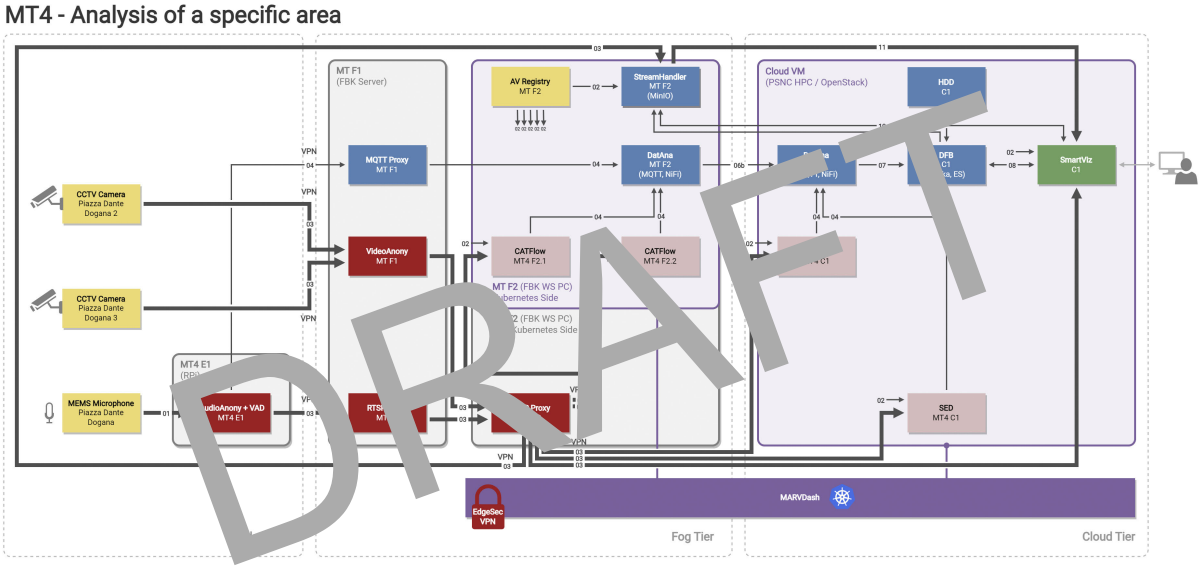


Figure 36. The DMP in the overall system architecture for use case MT4: Analysis of a specific area

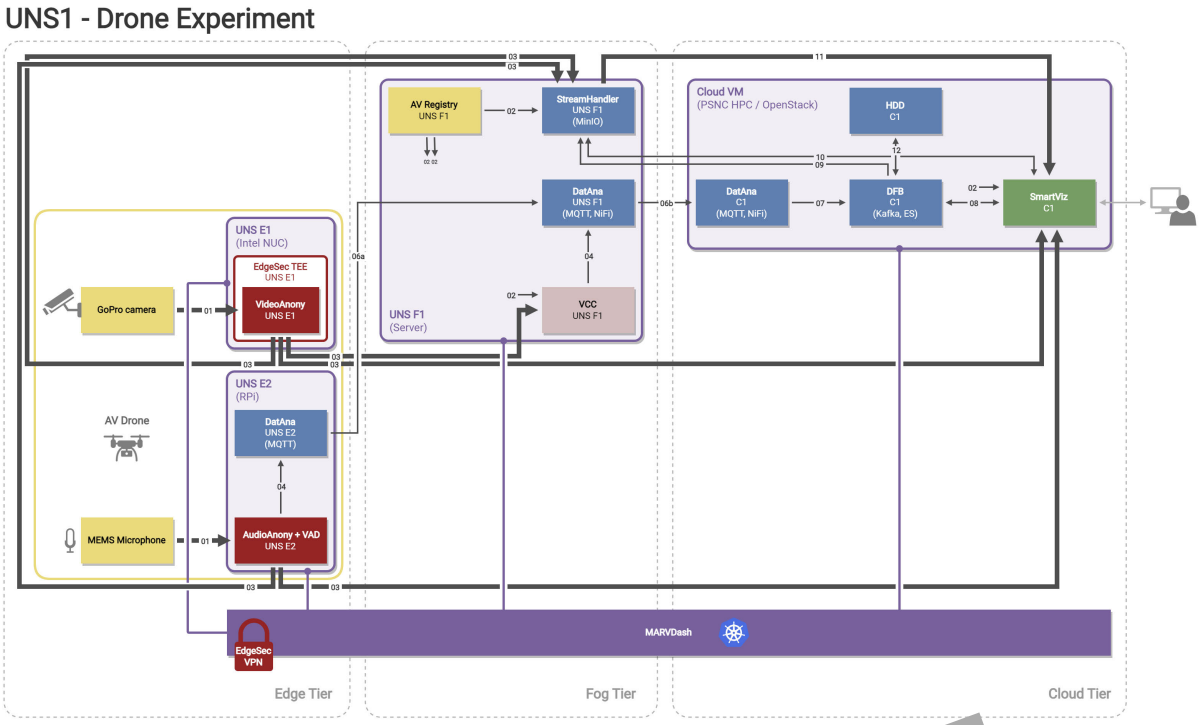


Figure 37. The DMP in the overall system architecture for use case UN[illegible]rone Experiment

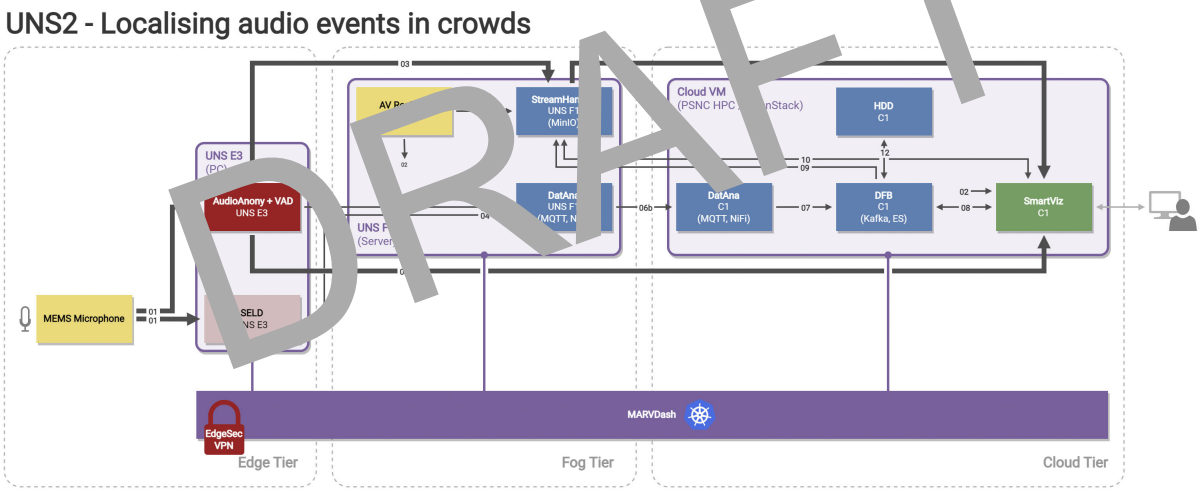


Figure 38. The DMP in the overall system architecture for use case UNS2: Localising audio events in crowds

3.2 AV data handling activities

MARVEL relies on the analysis of multimodal AV data to provide real-time inference results to its users. The AV data are typically produced by AV sources at the edge, i.e., microphone and camera devices that are connected to the internet and provide a live streaming service of the AV data. In this context, a thorough analysis was carried out to identify the live-streaming protocol that best fits the MARVEL project's scope from the available industry-standard protocols. The MARVEL Consortium nominated the RTSP as the protocol to be universally used for all needs of live streaming of AV data within the MARVEL framework. The standard for RTSP has been published as RFC236 in 1998 by the Multiparty Multimedia Session Control

Working Group (MMUSIC WG) of the Internet Engineering Task Force (IETF) in 1998[33]. According to the RFC236 standard, RTSP is an application-layer protocol for the setup and control of the delivery of data with real-time properties. RTSP provides an extensible framework to enable controlled, on-demand delivery of real-time data, such as audio and video. Sources of data can include both live data feeds and stored clips. This protocol is intended to control multiple data delivery sessions; provide a means for choosing delivery channels such as UDP, multicast UDP, and TCP; and provide a means for choosing delivery mechanisms based upon RTP. Even though it is not one of the most recent protocols for live AV streaming, RTSP is still widely used in the industry and allows backwards compatibility with the majority of commercial CCTV camera models, since it has been adopted by most manufacturers during the last decades. This is a very important aspect in the context of MARVEL, as its business case foresees the use of existing surveillance infrastructure of cities.

For the needs of MARVEL, the transmission of the anonymised live stream takes place in two stages: 1) the non-anonymised audio or video live stream from the IP cameras or microphones is read and processed by the VideoAnony/AudioAnony component, and 2) the anonymised audio or video stream is published via an RTSP[34] server so that other MARVEL services that need to consume live AV data streams (e.g., AI components) can establish a connection to the server and start consuming the anonymised live stream.

In the case of VideoAnony, it reads the RTSP streaming data from an IP camera using OpenCV[35] and then performs face and number plate detection and anonymisation using deep learning models. With the use of OpenCV and the GStreamer[36] the anonymised visual stream is sent to a server typically residing in the same processing layer.

A similar procedure to the VideoAnony applies to AudioAnony as well. However, the audio data are handled differently in the three pilots. In GRN and UNS, audio feeds are provided by either the cameras or the microphones as RTSP streams. Since in GRN anonymisation is not required, FFmpeg[37] is used to re-stream the audio to an RTSP server that then is accessible by all components. Conversely, in UNS, AudioAnony reads the audio stream and publishes the anonymised version on the retrieved RTSP server. In MT, microphones are installed on edge devices (designed and deployed specifically for MARVEL). Each device reads the data, anonymises the audio segments and publishes them using a local RTSP server (i.e., 1 server per microphone). Since the edge devices are outside the Kubernetes cluster, a re-streamer reads the streams from each RTSP server and publishes them on the Kubernetes RTSP server (see WP5 deliverables for more technical details).

Inside the Kubernetes cluster these streams are processed and/or consumed by various MARVEL components. Below two of the core components related to the advertisement and processing of the AV data are briefly presented. More information regarding these components can be found in the WP5 deliverables, such as D5.4 and D5.6 just to name a few.

- The AV Registry component is used to store and distribute information concerning AV sources. It exposes a REST API that is accessible to all services deployed in the MARVEL Kubernetes cluster to allow them to retrieve information related to available

[33] https://datatracker.ietf.org/doc/html/rfc2326

[34] https://github.com/aler9/rtsp-simple-server

[35] https://opencv.org/

[36] https://github.com/GStreamer/gstreamer

[37] https://ffmpeg.org/

AV sources in each MARVEL use case/deployment. The relative REST API foresees two calls that can be used to (1) retrieve all stored AV source documents and (2) retrieve a specific AV source document by referring to a particular AV source id. The REST API returns the requested information in the form of JSON documents that follow the specification of the Camera data model (Section 5.3.2.1 of D5.4).

- StreamHandler receives RTSP streams as input from the various AI components developed and deployed under the MARVEL framework. These streams are segmented into predefined time AV files and are stored in the storage component of StreamHandler. These AV files are sent to two other components, SmartViz and Data Corpus. SmartViz can access those AV files through a REST API endpoint by requesting an AV file for a specific time range and source. Then the URL of the newly created file is sent to the SmartViz component. Another way of interaction between SmartViz and StreamHandler is through the event API endpoint. The functionality is the same as with the previously mentioned endpoint, however in this case the AV file is created based on the events advertised from the DFB component and before it is requested from the SmartViz. Finally, StreamHandler periodically transmits the segmented files to the Data Corpus, by using a routine procedure that transmits the data over an HTTP channel to the Data Corpus platform.

3.3 The MARVdash facilitator

As already described in D2.2, MARVdash is a dashboard for Kubernetes that facilitates domain experts to interact with resources in the E2F2C MARVEL platform in an easy and user-friendly way. Instead of performing processing tasks itself, MARVdash offers the necessary mechanisms for efficient deployment. It utilises a service templating mechanism to configure and initiate services. Each service is characterised by a set of variables, and users have the ability to specify values for these variables as execution parameters via the dashboard prior to deployment. Additionally, MARVdash takes care of configuring other "internal" platform settings, including the location of a private docker registry, external DNS name, and other relevant configurations.

During the second development phase of the project, the private docker registry under MARVdash, which hosts the container images, was enriched with newer versions from partners as they continuously updated their code. Partners were able to run their new version of code by just redeploying a pre-existing template. This template had previously been created and referenced the image version as a variable, so there was no need to create a new template for each new image version.

Furthermore, with the latest version of MARVdash, the templates became even simpler by removing the tolerations and labelling part from the YAML. Those parts were replaced by a dropdown menu at the creation of the service thus providing an even simpler interaction with the user.

In the latest version of MARVdash, Grafana, Prometheus[38] and Loki[39] were included. Grafana provides a comprehensive range of features and functionalities to enhance the capabilities of a Kubernetes dashboard. As a powerful data visualisation and analytics platform, Grafana enables users to effectively monitor and analyse the performance and health of their Kubernetes cluster. Providing to MARVdash users a rich set of customisable dashboards, it allows users to create

[38] https://prometheus.io/

[39] https://grafana.com/oss/loki/

visually appealing representations of key metrics, such as CPU usage, memory utilisation, network traffic, and more. These dashboards can be tailored to specific requirements and provide real-time insights into the state of the Kubernetes environment. Additionally, Grafana offers advanced querying and filtering capabilities, empowering users to explore and drill down into their Kubernetes data with ease. It supports various data sources and integrates seamlessly with Prometheus, enabling users to leverage existing monitoring data for comprehensive visualisations. Overall, Grafana serves as a valuable tool for MARVdash users, enabling them to gain deeper insights, monitor performance, and make data-driven decisions for their Kubernetes clusters.

In addition to Grafana and Prometheus, Loki, which is a powerful log aggregation system, offers significant benefits to MARVdash. It enables efficient and centralised collection, indexing, and exploration of logs generated by various components within a cluster. With Loki, MARVdash users can easily search, visualise, and analyse log data from different Kubernetes pods, nodes, or namespaces. By integrating with popular logging libraries and frameworks, Loki seamlessly captures and indexes logs in a scalable and cost-effective manner. Its ability to store logs in a highly compressed and horizontally scalable manner significantly reduces storage costs and simplifies log management. Moreover, Loki supports advanced querying and filtering capabilities, allowing MARVdash users to extract meaningful insights from their logs. With Loki integrated into a MARVdash, users gain enhanced observability, troubleshooting capabilities, and the ability to identify and resolve issues quickly within the MARVEL framework.

All the components that are part of the DMP toolkit can be easily deployed and managed through MARVdash, leveraging the intuitive interface and simplified deployment mechanisms provided by the dashboard. More specifically, DatAna installations of Apache NiFi are deployed in many layers of the MARVEL E2F2C framework. These installations are simply instantiated through MARVdash, by selecting the appropriate layer through the taint and affinity Kubernetes methods that were described in D2.2. Likewise, Mosquitto MQTT message brokers, which have been included in MARVEL to decouple messaging among components and DatAna, can be easily deployed through MARVdash to the appropriate nodes of the E2F2C deployment. In addition to the above, NiFiRegistry, which helps in data flows version control, is instantiated through MARVdash in the MARVEL cloud.

DFB is similarly deployed through MARVdash to the cloud. All the components of the DFB such as Kafka, Elastic Search, Kibana[40], and the rest are deployed to Kubernetes Cluster by the instantiation of the appropriate MARVdash templates leveraging the intuitive interface and simplified deployment mechanisms provided by MARVdash. StreamHandler, which is parted of different components is also deployed through MARVdash in the fog layer. Likewise, HDD is instantiated to the cluster through MARVdash and deployed to the cloud layer. MARVdash makes it effortless to deploy and manage applications. With its intuitive design, the dashboard allows users to easily navigate through various components, view pod statuses, monitor resource utilisation, and analyse logs and metrics. Additionally, MARVdash makes it incredibly easy to update applications deployed within the cluster. MARVdash users can quickly roll out new versions, or modify configuration parameters with just a few clicks, ensuring the cluster remains up-to-date and responsive to evolving needs.

MARVdash as previously described in D2.2 enhances also the deployment of the Data Corpus by integrating a web proxy service. This proxy implementation enables secure external access to the Data Corpus API services. Moreover, a new ingress endpoint is registered within

[40] https://www.elastic.co/kibana/

Kubernetes, ensuring an HTTPS-compliant URL that guarantees the encryption of data exchanges. Additionally, the endpoint is configured to utilise the existing authentication mechanism provided by MARVdash, further enhancing the security and authentication process. The Data Corpus, in the updated version of MARVdash, is accessed directly from the web page menu. Furthermore, a new proxy service is added in the Kubernetes Cluster in order to download files from the Hadoop[41] cluster.

[41] https://hadoop.apache.org/

4 Future plans after M30 of the project

4.1 DatAna

DatAna is considered finalised for the MARVEL use cases after this M30 release. The upcoming months will be related to bug fixing and minor improvements in the data flows, if required by the feedback from the pilots. DatAna will also provide input to WP5 in relation to the benchmarking activities and provide the necessary support for the preparation of demos and related activities.

4.2 DFB

In the final part of the project after this M30 release and beyond the project lifetime, the following activities are being considered for the further development and improvement of the DFB solution within the context of MARVEL:

- Further test and refine the data fusion process of inference results.
- Prepare additional meaningful visualisations in the DFB Kibana tool for long-term analytics.
- Investigate and analyse scenarios of (a) interfacing with third-party components that are external to MARVEL, (b) potential interfacing with multiple DatAna and SmartViz instances and (c) improved scalability offered by HDD, in the context of T5.5 activities and in the direction of demonstrating the capacity of the MARVEL framework to achieve extreme scale analytics.
- Conduct more rigorous testing and measure performance more accurately based on diversified load scenarios.

4.3 StreamHandler

INTRA's StreamHandler recently received a complete refactoring both from architectural and code perspective. This release aims to increase the efficiency and resilience of the component. Another major feature incorporated was the use of DFB's event advertisement for compiling AV files. Considering these, within the MARVEL framework, StreamHandler is finalised by meeting all the initially stated requirements and specifications. The period left till the end of the project will be devoted to minor tweaks to further boost the efficiency and performance of StreamHandler on each Pilot use case. More details regarding the performance enhancement of the StreamHandler will be reported in deliverable D5.5.

4.4 HDD

The new results have highlighted some qualitative differences with respect to the conclusions obtained with the MARVEL-tailored simulation model of D2.2, which suggested the need for cross-checking analysis with real-life scalable data before production deployment. Therefore, our immediate plan for the short-term future is to explore the scalability potential of HDD, also in the scope of WP5.

5 Conclusion

The completion of Deliverable D2.4 represents a significant advancement in the development and evolution of the MARVEL Management and Distribution Toolkit. Building upon the foundation established in D2.2, this follow-up deliverable has effectively addressed stakeholder feedback and requirements, resulting in a more refined and robust toolkit.

The enhancements introduced in D2.4 align with the project's overarching objectives of optimising AV data management and maximising the value derived from MARVEL components. By expanding functionality, improving interconnection interfaces, enhancing data governance, and optimising performance, a comprehensive solution has been achieved.

This document has provided comprehensive information about the development of the DMT since the inception of the project's technical development, with a particular focus on the final stages of the project (M30). It has covered the main components of the DMT developed within the MARVEL framework, ensuring that all necessary details are provided.

Lastly, the partners have outlined their plans for the future of their developed components beyond the completion of the MARVEL project. These plans primarily involve optimising their components and exploring new exploitation scenarios to further enhance their capabilities.

Overall, Deliverable D2.4 signifies a significant milestone in the advancement of the MARVEL Management and Distribution Toolkit, providing a solid foundation for future development and exploitation.