European Commission | Horizon 2020 European Union funding for Research & Innovation

Big Data technologies and extreme-scale analytics

MARVEL

Multimodal Extreme Scale Data Analytics for Smart Cities Environments

D5.6: MARVEL Integrated framework – final version [†]

Abstract: This deliverable showcases the **final version of the MARVEL Integrated framework** (**R2**), as well as all the underlying integration activities carried out towards its realisation. R2 is designed to address ten use cases across the three MARVEL pilots. The deliverable first presents the plan and methodology that were followed for the integration of MARVEL components into R2, followed by a description of these components and their role. Subsequently, a detailed presentation of the design and specifications of the MARVEL R2 framework and its application in the use cases is provided, along with the associated I/O interfaces, data models and UI/UX, while giving an account of the employed infrastructure. Then, the deliverable discusses the main challenges and issues encountered, presents the main achievements of R2 and its contribution to the overall MARVEL goals, and ends with the next steps and a summary of the conclusions.

Contractual Date of Delivery	30/06/2023
Actual Date of Delivery	21/07/2023
Deliverable Security Class	Public
Editor	*Tassos Kanellos (ITML)*
Contributors	ITML, FORTH, IFAG, AU, ATOS, CNR, INTRA, FBK, AUD, TAU, STS, MT, UNS, GRN, ZELUS, PSNC
Quality Assurance	*Ilias Seitanidis (INTRA)* *Claudio Cicconetti (CNR)*

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

The *MARVEL* Consortium

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

Document Revisions & Quality Assurance

Internal Reviewers

1. *Ilias Seitanidis (INTRA)*
2. *Claudio Cicconetti (CNR)*

Revisions

Version	Date	By	Overview
1.0.0	21/07/2023	Tassos Kanellos (ITML)	Final version ready for submission to the EC
0.8.0	20/07/2023	Despina Kopanaki (FORTH)	Final version approved by the PC
0.7.0	11/07/2023	Tassos Kanellos (ITML)	Final draft approved by internal reviewers
0.6.0	10/07/2023	Tassos Kanellos (ITML)	Pre-final draft for final approval by internal reviewers
0.5.5	06/07/2023	Ilias Seitanidis (INTRA) Claudio Cicconetti (CNR)	Comments on complete draft
0.5.0	03/07/2023	Tassos Kanellos (ITML)	Complete draft released for internal review
0.4.0	12/06/2022	ITML, FORTH, IFAG, AU, ATOS, CNR, INTRA, FBK, AUD, TAU, STS, MT, UNS, GRN, ZELUS, PSNC	Updated contributions integrated
0.3.0	29/05/2022	ITML, FORTH, IFAG, AU, ATOS, CNR, INTRA, FBK, AUD, TAU, STS, MT, UNS, GRN, ZELUS, PSNC	Initial contributions integrated
0.2.0	05/05/2023	Tassos Kanellos (ITML)	Revised ToC
0.1.5	02/05/2023	Manolis Falelakis (INTRA) Dragana Bajovic (UNS)	Comments on ToC
0.1.0	21/04/2023	Tassos Kanellos	Initial ToC.

Disclaimer

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

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

Table of Contents

List of Tables

List of Figures

List of Abbreviations

AAC	Automated Audio Captioning
AI	Artificial Intelligence
ADC	Analog-to-digital converter
ALSA	Advanced Linux Sound Architecture
ASIC	Application-specific integrated circuit
API	Application Programming Interface
AT	Audio Tagging
AV	Audio-Visual
AVAD	Audio Visual Anomaly Detection
AVCC	Audio Visual Crowd Counting
CCTV	Closed-circuit television
CI/CD	Continuous Integration / Continuous Delivery
CPU	Central processing unit
dBSPL	Decibel Sound Pressure Level
DFB	Data Fusion Bus
DMP	Data Management Platform
DMT	Decision-Making Toolkit
DNS	Domain Name System
E2E	End-To-End
E2F2C	Edge-to-Fog-to-Cloud
EC	European Commission
ELK	Elasticsearch Logstash Kibana
FFMPEG	Fast Forward MPEG
GAN	Generative adversarial network
GUI	Graphical User Interface
HDD	Hierarchical Data Distribution
HDFS	Hadoop Distributed Files System
HLS	HTTP Live Streaming
HPC	High Performance Computing
HTTPS	Hypertext Transfer Protocol Secure
ICT	Information and communications technology
I/O	Input/Output
IoT	Internet of Things

ISO	International Organisation for Standardisation
ITS	Issue Tracking System
IP	Internet Protocol
JSON	JavaScript Object Notation
KPI	Key Performance Indicator
LPC	Linear predictive coding
LSTM	Long Short-Term Memory
MEMS	Micro Electro-Mechanical Systems
ML	Machine Learning
MQTT	Message Queuing Telemetry Transport
MVP	Minimum Viable Product
NAT	Network address translation
NUC	Next Unit of Computing
OLAP	Online Analytical Processing
OS	Operating system
PDM	Pulse Density Modulation
PFLOPS	Peta Floating-Point Operations Per Second
POE	Power Over Ethernet
QA	Quality Assurance
RAM	Random Access Memory
RBAD	Rule-Based Anomaly Detection
REST	Representational State Transfer
RTSP	Real-Time Streaming Protocol
R1	1st (initial) Release of the MARVEL Integrated framework
R2	2nd (final) Release of the MARVEL Integrated framework
SCONE	Secure Container Environment
SDK	Software Development Kit
SDM	Smart Data Models
SED	Sound Event Detection
SELD	Sound Event Localisation and Detection
SGX	Software Guard Extensions
SLURM	Simple Linux Utility for Resource Management
SNR	Signal-to-Noise Ratio
SSH	Secure Socket Shell

SSO	Single Sign-On
TAD	Text Anomaly Detection
TB	Terabyte
TCP	Transmission Control Protocol
TEE	Trusted Execution Environment
TRL	Technology Readiness Level
UAV	Unmanned Aerial Vehicle
UC	Use Cases
UDP	User Datagram Protocol
UI	User Interface
UJ	User Journey
URL	Uniform Resource Locator
USB	Universal Serial Bus
UTC	Coordinated Universal Time
UX	User experience
VAD	Voice Activity Detection
VCC	Visual Crowd Counting
VCS	Version Control System
VGG	Visual Geometry Group
ViAD	Visual Anomaly Detection
VM	Virtual Machine
VPN	Virtual Private Network
WP	Work Package
WS	Workstation
YOLO	You Only Look Once (Real-time object detector)
YOLO-SED	You Only Look Once – Sound Event Detection

Executive Summary

This deliverable showcases the **final version of the MARVEL Integrated framework** (also referred to as **2nd Release** or **R2**), as well as all underlying integration activities carried out towards its realisation. Following the release of the **MARVEL Minimum Viable Product (MVP)** in M12 and the **initial version of the MARVEL Integrated framework (R1)** in M18, the MARVEL project aims for a final release of a complete, end-to-end demonstrator that can operate under real-life conditions, serving the purpose of a technical validation of the overarching goals of MARVEL and displaying the potentials of the proposed solution. The deliverable has been developed in the context of Task T5.3 '*Continuous integration towards MARVEL's framework realisation*' within WP5 '*Infrastructure Management and Integration*', under Grant Agreement No. 957337.

The work presented in this report embarks from previously submitted deliverables D1.2 '*MARVEL's Experimental protocol*' and D1.3 '*Architecture definition for MARVEL framework*', which define in detail the MARVEL pilot use cases and the refined framework architecture, respectively. It also builds upon the former MARVEL MVP release, which was presented in the submitted deliverable D5.1 '*MARVEL Minimum Viable Product*' and can be considered as an update to the initial version of the MARVEL Integrated framework (R1), presented in the submitted deliverable D5.4 '*MARVEL Integrated framework – initial version*'. This preceding work forms the foundations for the development of R2.

Ten use cases are addressed by R2 in the three MARVEL pilots (Malta, Trento, Novi Sad), being the complete set of foreseen MARVEL use cases and comprising five new use cases that were introduced after R1. A detailed plan was constructed at the outset of the R2 integration activities that led the mapping of components to the new use cases, as well as the design and specification of the architecture, interfaces and data models to be implemented, and eventually the deployment, demonstration and validation of R2. R2 marks a considerable progress beyond the MVP and R1, delivering the full set of MARVEL components seamlessly integrated and deployed across all layers of the E2F2C continuum to provide an efficient system that can operate under real-life conditions and address the selected use cases while demonstrating the added value of MARVEL.

This report provides a comprehensive documentation of all the participating components and technologies offered by the MARVEL partners, including functionality and role of each component within R2. The report also covers a complete documentation of all interfaces, APIs and data models that facilitate communication and integration between components, the architecture and implementation details of the R2 framework in the addressed use cases, a description of the main UI/UX design and a detailed presentation of the allocated infrastructure that supports the implementation of R2.

In addition, this deliverable discusses the main challenges and issues that were encountered during integration activities and how they were addressed. Furthermore, this report presents the main achievement of R2 and how R2 contributes to specific WP5 and overall project goals, and its function as a solid foundation for a future, large-scale deployment and operation of MARVEL in real-world settings after the lifetime of the project. Finally, the future work being considered is presented, followed by the main conclusions of this report.

1 Introduction

1.1 Purpose and scope of this document

The purpose of this deliverable is to describe the scope, design rationale, technical details, and integration activities for the **final version of the MARVEL Integrated framework** (also referred to as **2nd Release** or **R2**). Within the context of the MARVEL project, R2 is the final release of the integrated framework, achieving a complete operational status and addressing all ten foreseen use cases, while offering new features and achieving improvements in comparison to R1 in terms of system stability, consistency, performance. R2 serves final validation purposes for the project's main objectives, offering a functional and fully operational demonstration in terms of end-to-end integration and delivery of value to the end-user.

In terms of design rationale, technical details, and integration activities, this deliverable explains how the MARVEL Consortium has selected representative use cases that connect all layers of the MARVEL architecture and mapped a series of components to them in order to demonstrate coherent instantiations of MARVEL as an integrated operational solution that collects, manages, processes, transfers, and visualises data across all architectural layers in an efficient pipeline to provide meaningful information to the end-user. A parallel objective was to demonstrate the versatility and adaptability of MARVEL for addressing diversified cases and complying with varying requirements and available infrastructure resources.

To complement the design and technical activities, this deliverable also lists the WP5 and project objectives that R2 addresses (Sections 1.3 and 7.2) and how it serves as a steppingstone towards the release of real-world and large-scale version of the MARVEL framework for a long-term sustainability of the results after the end of the project.

1.2 Intended readership

Deliverable D5.6 *'MARVEL Integrated framework – final version'* is a public document that accompanies the public demonstrator of the MARVEL framework. This document is addressed to all relevant stakeholders, potential users of the framework and communities working in the fields of IoT, AI, and Smart Cities in order to show the feasibility of the MARVEL premises, the validity of the services provided, and a comprehensive documentation of the technical solution and the methodology that was implemented to develop it.

This report also serves the purpose of orienting parties interested in the technical aspects of MARVEL to the entire MARVEL technical documentation. Since D5.6 is meant to comprehensively present the entire MARVEL framework, its constituent components and internal operation, it can act as an introductory document that can lead the reader to more specialised reports that refer to specific subsystems, components or other technical aspects of MARVEL in more detail.

Additionally, this document is expected to serve as a reference to any activities related to future integrated releases of the MARVEL framework that will build upon R2 and potentially comprise larger-scale implementations, incorporating additional features, introducing performance and stability improvements and addressing new use cases.

1.3 Contribution to WP5 and project objectives

This deliverable has been composed within the context of WP5 *'Infrastructure Management and Integration'*, and more specifically, it constitutes the second major output of Task 5.3 *'Continuous integration towards MARVEL's framework realisation'*. The Description of Action (DoA) states the objectives of WP5 as such:

> *WP5 main objective is to ensure successful E2F2C framework delivery for distributed extreme-scale audio analytics. The framework allows for powerful, scalable, and real-time processing of multimodal audio-visual data on top of distributed deployment of MARVEL ML models. In detail, WP5 aims to ensure:*
>
> *(i) provision and configuration of a powerful HPC infrastructure;*
> *(ii) orchestration of infrastructure resource management and optimised automatic usage of external computational and storage resources;*
> *(iii) seamless integration and quality assurance of software releases;*
> *(iv) quantifiable progress against societal, academic and industry validated benchmarks;*
> *(v) real-life experimentation and validation at large scale; and*
> *(vi) continuous alignment with the responsible AI planning and guidelines set out in T1.3.*

The **final version of the MARVEL Integrated framework** (also referred to as **2nd Release** or **R2**) constitutes an important milestone towards achieving the objectives of this work package, by providing a complete, operational version of the envisioned framework that addresses and incorporates all mentioned attributes of this framework, including (i) provision of the required infrastructure (Section 5.6), (ii) infrastructure resource management (Section 4.6.2) and optimised automatic usage of external computational and storage resources (Section 4.5), (iii) integration of offered technologies (Section 5), (iv) tools for monitoring and measuring performance (Section 3.8.3), (v) experimentation and validation under operational conditions (Section 3.8), (vi) updated approach regarding responsible AI planning (Sections 4.2, 4.7, 5.4.4-5.4.7).

Specifically, for Task T5.3, the DoA states that:

> Task 5.3 *will implement and deploy the MARVEL integrated framework that realises the technology convergence defined in the MARVEL architecture specification T1.4. The MARVEL E2F2C Framework bridges Big Data technologies to IoT, Edge/Fog/Cloud computing and HPC, to allow real-time decision making and monitoring of multimodal smart cities environments. This task realises societal and industrial opportunities in the smart-city domain (T6.2) by ensuring a smooth and effective integration of the separate MARVEL components. Aspects like interoperability, scalability, accountability, transparency, responsibility, and performance will be considered, as well as a continuous delivery approach, to achieve quality assurance in software releases. Releasing software frequently will not only respect the natural evolution of the technologies developed within the project but it will also allow developers to react promptly to the continuous data providers feedback. Reliability of these developments is also increased following this strategy as recurrent releases usually involve lesser and harmless changes. According to the plan (Sect.3.1.1), this task will describe in detail the MARVEL releases namely a proof-of-concept demonstration (Minimum Viable Product (MVP)) - M12, the 1st complete prototype – M18 and the 2nd Final prototype – M30.*

This deliverable describes R2, which is the third output of task T5.3. R2 builds upon the MVP (D5.1[1]) and R1 (D5.4[2]) to establish the final complete and integrated version of the MARVEL framework that operates under real-life conditions and addresses all the fundamental notions

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

[2] "D5.4: MARVEL Integrated framework – initial version" Project MARVEL, 2022. Not publicly released (confidential)

mentioned above, paving the way for future large-scale deployments of the MARVEL framework. R2 marks the achievement of considerable progress beyond R1 by incorporating additional technologies, addressing additional use cases, improving stability, deploying on extended infrastructure and adding new input/output devices (sensors/actuators).

R2 is based on a revised approach towards the convergence of diverse technologies, effectively integrated under a common scheme that is based on a reference 'AI inference pipeline' architecture, coupled with comprehensive APIs, I/O interfaces and data model specifications that ensure efficient communications between multiple component endpoints.

R2 delivers a demonstration of real-time interaction with the end-user under real-life operational conditions, provided by the MARVEL pilots. R2 design and development addresses issues of interoperability, scalability (Sections 4.3.4, 5.1, 5.2, 5.3), accountability, transparency, responsibility (Section 3.5, 4.5), and performance (Sections 4.5, 5.4, 5.4.11, 3.8.3) and has been based on a continuous integration/continuous delivery (CI/CD) approach (Section 3.7).

1.4 Relation to other WPs and deliverables

This deliverable relies on the foundational work conducted within WP1 *'Setting the scene: Project setup'*. More specifically, the selection of use cases for demonstration draws from the detailed material on Use Case descriptions of deliverable D1.2 *'MARVEL's Experimental protocol*'. Additionally, deliverable D1.3 *'Architecture definition for MARVEL framework'*[3] is an important source for this work, as it contains the refined architecture, which is the blueprint for this release, as well as subsequent releases. D1.3 also provides useful information that D5.6 builds upon, as for example the description of the available MARVEL components and their TRL, the grouping of components into building blocks (Subsystems according to their functional role in the platform), and the outline of integration processes that need to be applied.

This deliverable is also coupled with work in progress within the context of WP2 *'MARVEL multimodal data Corpus-as-a-Service for smart cities',* WP3 '*AI-based distributed algorithms for multimodal perception and situational awareness',* and WP4 *'MARVEL E2F2C distributed ubiquitous computing framework'*. More specifically:

- From WP2, integral parts of R2 are the Data management and distribution platform (T2.2), and the MARVEL Corpus-as-a-Service (T2.4).
- From WP3, R2 includes AI-based methods for audio-visual data privacy (T3.1), AI algorithms for Audio-Visual (AV) intelligence (T3.3), federated learning approaches (T3.2), adaptive E2F2C distribution and optimisation of AI tasks (T3.4) and Edge-optimal ML/DL deployment (T3.5).
- From WP4, draws outputs from the optimised audio capturing (T4.1), audio anonymisation (T4.2), security in the complete E2F2C (T4.3) and the Decision-Making Toolkit (T4.4).

Within WP5, there has been a close collaboration with T5.1 and T5.2, with regards to the underlying infrastructure and resource management, respectively. Additionally, there is a close connection to T5.4 that aims to set the benchmarks to validate R2 results.

D5.6 is an extension of D5.1 *'MARVEL Minimum Viable Product'*, which set the stage for the subsequent releases and especially of D5.4 *'MARVEL Integrated framework – initial version'*, which formed the main foundation on top of which D5.6 was developed. Specifically, R2 of

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

D5.6 builds upon the design, development and integration approach established for the MVP of D5.1 and for R1 of D5.4 as well as on the lessons learnt from them.

D5.4 was marked as a **confidential** deliverable and referred to an **initial version** of the MARVEL integrated framework. **D5.6** is marked as a **public** deliverable and aims to present the complete and **final version** of the MARVEL integrated framework. Therefore, information from D5.4 is partially re-used in D5.6 and revised where necessary, while new information introduced during R2 activities is appended.

This deliverable will also serve as a reference to the upcoming deliverable D5.7 *'MARVEL's framework large scale deployment'* that will describe the possibilities of the application of the MARVEL integrated framework in large-scale deployments. The R2 demonstrator will provide the data for the evaluation that will be part of D5.5 *'Technical evaluation and progress against benchmarks'*.

Finally, a considerable part of the work for WP6 '*Real-life societal experiments in smart cities environment*' refers to information contained in this deliverable and most notably deliverable D6.3 *'Demonstrators execution – final version'*.

1.5 Structure of the document

The structure of this document is as follows:

- *Section 1* introduces the reader to this report and links the related work to the overall project context.
- *Section 2* provides an **overview of R2 and its objectives** as well as the **use cases** that R2 addresses**.**
- *Section 3* discusses the **plan and methodology for integration** activities that have been applied for the development of the final integrated prototype of the MARVEL framework (R2).
- *Section 4* provides a comprehensive **list of MARVEL components** that have been integrated in the use cases of R2.
- *Section 5* presents the **Design and Specifications of R2**, including the **architecture**, the **I/O interfaces (APIs)**, the **Data Models**, the **UI/UX** design, and the **infrastructure**.
- *Section 6* describes the **main challenges and issues encountered** throughout the R2 integration activities as well as **the ways in which they were resolved**.
- *Section 7* links the results presented in previous sections to the **overall MARVEL goals and objectives** and presents the **main R2 achievements**.
- *Section 8* refers to **plans for the further development** of the MARVEL Integrated framework in the future.
- *Section 9* summarises and concludes this document.

2 R2 Overview and Use Cases

In this section, we set the scope and goals of the **MARVEL integrated framework** considering both releases, i.e., the **initial version** (**1st Release – R1**) and the **final version** (**2nd Release – R2**). An overview of the integrated framework and its objectives is provided, followed by a description of the MARVEL use cases, and the rationale behind their implementation.

2.1 R2 Overview and Objectives

The MARVEL project adopted a highly practical and systematic approach to release a system that is demonstrated and validated in real-life operational environments of Smart Cities. Towards this final goal, the **R2 milestone**, MS7 *'MARVEL integrated version (2nd release)'*, is set to ensure the convergence and seamless integration of MARVEL subsystems and components as well as the successful framework operation within the selected MARVEL use cases, building upon the achievements of the **R1 milestone**, MS3 *'MARVEL Minimum Viable Product (MVP)'* on M12 and of the **MVP milestone** MS4 *'MARVEL integrated version (1st release)'* on M18.

The **MVP** release (D5.1) served as the groundwork for all subsequent development and integration activities. Even though the MVP focused on a limited part of the overall MARVEL framework (e.g., one use case from one pilot addressed, involvement of a small subset of MARVEL components, limited infrastructure, partial automation, offline input data), it proved beneficial in streamlining the methodological approach for the integration activities. The MVP was also crucial for identifying the main next steps and challenges. The MVP managed to successfully achieve its goals and delivered a functional product that demonstrated the value of the MARVEL framework in the selected use case.

Following the MVP release and the lessons learnt from it, there was a phase of re-evaluation of the approach that would be adopted for the development and integration of the initial integrated prototype of the MARVEL framework (**R1**). In R1, the complexity significantly increased, as a larger subset of components was integrated, five use cases were addressed, infrastructure resources from three pilot sites were used and a comprehensive system architecture design was established, including the specification and documentation of necessary APIs and data models. In R1, live input data were used, security and privacy protection measures were taken, data management was organised, and more thorough integration and deployment methods and practices were employed. R1 delivered an initial integrated framework that was successfully tested and demonstrated, meeting its goals for the addressed use cases.

R2 builds upon the MVP (D5.1[4]) and R1 (D5.4[5]) to establish the final complete and integrated version of the MARVEL framework that operates under real-life conditions and addresses all the fundamental notions mentioned above, paving the way for large-scale deployments of the MARVEL framework. Specifically, the R2 objectives are as follows:

- Implement the MARVEL framework in all the **ten (10) foreseen use cases** in the three (3) MARVEL pilots.
- Integrate in a seamless manner **all the MARVEL technological assets** (Section 4), as defined in D1.3. In R2, the complete set of the MARVEL components needs to be

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

[5] "D5.4: MARVEL Integrated framework – initial version" Project MARVEL, 2022. Not publicly released (confidential).

represented across the addressed R2 use cases and needs to be carefully mapped to use cases, E/F/C layers and infrastructure nodes.

- Define the component and framework configurations according to the needs of the use cases, demonstrating the **versatility of MARVEL to adapt to diverse use cases, while maintaining a consistent and coherent architectural approach** that can be applied universally.
- Showcase scalable and real-time efficient **management and processing of live feeds of multimodal audio-visual data**.
- Achieve a **distributed E2F2C deployment of ML models** and other relevant technologies, ensuring **access to data** and maintaining **privacy preservation and alignment with the responsible AI guidelines** as set out in D1.2.
- **Demonstrate configuration, orchestration and automated provision of the infrastructure**.
- **Deploy and successfully operate the framework on the target infrastructure** of R2 use cases, involving multiple nodes and under a unified deployment environment (Kubernetes cluster).
- **Demonstrate real-time decision-making and monitoring** of multimodal smart city environments within each of the R2 use cases.
- **Implement and extend CI/CD processes** that were established for R1.
- Configure the R2 framework to enable its **benchmarking**.
- Implement **performance improvements** and more efficient use of computational resources in R2.
- Add features for improving **centralised system monitoring** including resource consumption.
- **Improve and expand R2 computational infrastructure** beyond the one used for R1.
- Support **additional AV sources and actuator devices** in R2.
- **Shift computation closer to the edge** in new use cases introduced for R2.
- Implement overall **system stability improvements** and increase resilience to AV stream intermittency and content corruption.
- Ensure **timely and successful component deployment, integration, and testing.**

The design, development and integration of R2 was driven, first and foremost, by the use cases it was called to address, presented in the next Section 2.2.

The delivery of R2 was achieved through meticulous planning and depended on the organisational mechanisms that were established, namely the detailed integration time plan (see Section 3.1.1), the realisation of technical integration meetings (see Section 3.1.2), the establishment of an Integration Board (see Section 3.1.3) and the implementation of other support tools, mechanisms and methods described in Section 3 (e.g., version control system, issue tracking system, specification documentation, infrastructure sizing).

An up-to-date description of the technological components that were integrated in R2 is presented in Section 4.

The complete design and specifications of the R2 release are presented in Section 5.

R2 was successfully delivered, following the deployment of the MARVEL components under a unified environment (Kubernetes cluster), bridging multiple infrastructure nodes that were available across the three pilots with the help of the MARVdash tool. R2 was thoroughly tested following planned Quality Assurance procedures (Section 3.8).

R2 is the final milestone in the development and delivery of the overall MARVEL framework. in the context of the project.

2.2 R2 Use cases

As described in detail in deliverable D1.2 *'MARVEL's Experimental protocol'*, there are three real-life experiments designed to be implemented and executed in pilots for three cities, Malta (GRN pilot), Trento (MT pilot) and Novi Sad (UNS pilot). For the first two, eight (8) different use cases are defined to support the real-life experiments, while for the latter pilot, two (2) use cases are defined. These use cases cover a wide range of activities to be monitored and potential events to be detected, analysed, and mitigated.

In the context of the **MVP**, a **single use case was selected** from the GRN pilot, namely *use case 'GRN4 – Junction Traffic Trajectory Collection'*. The delivery of the MVP successfully addressed this use case on an initial level and set a significant precedent that paved the way for subsequent development and integration activities.

In the context of **R1**, **five (5) use cases were selected to be addressed**. The selection process for the R1 leveraged the experience gained from the MVP. It followed the same principles that were established in the MVP but was also further elaborated. More specifically, the selection process was based on several criteria that were discussed during the pilot-focused meetings, including (i) maturity of the use cases, (ii) infrastructure availability, (iii) data availability, (iv) feasibility of completion within the given timeframe, (v) readiness of technological components that would be needed and (vi) demonstrating diversity among the chosen use cases.

The result of this process was the selection of the following five use cases for R1:

- *use case 'GRN3 – Traffic Conditions and Anomalous Events'* of the Malta (GRN) pilot.
- *use case 'GRN4 – Junction Traffic Trajectory Collection'* of the Malta (GRN) pilot.
- *use case 'MT1 – Monitoring of crowded areas'* of the Trento (MT) pilot.
- *use case 'MT3 – Monitoring of parking places'* of the Trento (MT) pilot.
- *use case 'UNS1 – Drone experiment'* of the Novi Sad (UNS) pilot.

R2 sets out to address the complete set of the ten (10) foreseen MARVEL use cases, continuing to be applicable for the five use cases that were introduced in R1 and covering five (5) new use cases that are introduced in R2:

- *use case 'GRN1 – Safer Roads'* of the Malta (GRN) pilot.
- *use case 'GRN2 – Road user behaviour'* of the Malta (GRN) pilot.
- *use case 'MT2 – Detecting criminal/anti-social behaviours'* of the Trento (MT) pilot.
- *use case 'MT4 – Analysis of a specific area'* of the Trento (MT) pilot.
- *use case 'UNS2 – Localising audio events in crowds'* of the Novi Sad (UNS) pilot.

R2 demonstrates a wider spectrum of possibilities that can be enabled by MARVEL through the ten addressed use cases. In this context, a wide variety of inputs and outputs is offered that can be managed and processed by R2, including vehicle and pedestrian identification and trajectories, sound event detection, characterisation and localisation, voice activity detection, crowd counting, anomaly detection, control of on-site devices (actuators), inference result post-processing, audio and video anonymisation and enhanced security. R2 showcases the full potential of MARVEL's AI functionalities while providing the end-user with an immediate view of the events and current state happening at the street level.

The R2 use cases expose all the most critical technical challenges in terms of orchestration and integration of components and demonstrate the variability and versatility of the MARVEL framework for meeting diverse requirements. The use cases are a very representative sample of real-world scenarios that can be brought forward by Smart Cities.

2.2.1 GRN Use Case 1 – Safer roads

This use case addresses the need to increase safety on urban roads for vulnerable road users, with the aim of encouraging the uptake of active travel modes in Malta. More specifically, this use case targets cycling and walking. Malta has witnessed a significant effort, from both the authorities and the bicycle commuting lobby, in encouraging cycling and walking, mainly through infrastructural changes. This use case takes this effort further and aims at detecting cyclists, including e-bikes and pedestrians, exiting a junction and alert car and motorised-vehicle drivers of their presence via variable message signs in the hope that car drivers take greater care and concentrate more in such circumstances.

In addition, detecting cyclists is a particularly interesting task in low visibility conditions because it is both more dangerous for these entities and more challenging from a technology point of view.

From an AI task point of view, this use case requires detectors for traffic entities (typically cycles and pedestrians) that are typically present at a junction. It is also necessary to resolve the exit carriageway taken by the vulnerable road users such that the respective message boards on that carriageway are triggered whilst avoiding false positives, the occurrence of which can reduce the system’s impact in the long term. This can be achieved by sampling the entrance to the road. However, it will also increase the system response time.

The detection and classification of entities are typically implemented using computer vision techniques. Detecting the cyclist is a known hard problem, and intuitively, the addition of the audio signal would not help. However, sound cues may potentially disambiguate a bicycle from a motorcycle or moped. In addition, audio-visual models may differentiate between bicycles and motorised bicycles, which is also a desired function.

These objectives will be attained with a YOLO-SED model. To simplify the YOLO implementation, CATFlow weights and parameters are used, and SED is applied to differentiate between motorbikes and bicycles. To minimise latency, YOLO-SED were implemented on the Jetson edge device to enable the installation of the edge device closer to the camera and associated sign point. An LED array driven by an Arduino board was used to replace the variable message sign intended to alert drivers. Messages from the detector are also forwarded to the UI to allow remote monitoring of junctions.

This use case should contribute towards an increase in the perceived safety on the roads and will therefore encourage commuters to consider cycling as an alternative mode of transportation. To determine the impact of this use case, surveys to gauge citizens' perceptions of safety with this device will be conducted.

2.2.2 GRN Use Case 2 – Road user behaviour

This use case addresses the need to monitor the behaviour of road users at a junction. This use case demonstrates tools that are useful in law enforcement and education campaigns targeting responsible driving, cycling, and other uses of the roads. Malta has experienced fast changes in the transport landscape to which human response often lags behind technical progress. Educational campaigns are one way to close this gap and have demonstrated their effectiveness in the past. This use case involves the classification of actions into a spectrum of examples demonstrating good to bad behaviour. This use case will not be implementing the latter campaigns or policies; however, it could be tried in different places and its output could be observed. Surveys will be used to find how this tool will be able to help local authorities.

Examples of actions include the way pedestrians cross over the intended crossings, whether cyclists dismount at pedestrian crossings, and whether car drivers stop in the delineated zone at junctions. The system will be able to count the number of times bad behaviour is detected before and after the execution of education campaigns or policy changes.

This use case uses the TAD component to detect anomalous trajectories and a rule-based anomaly detector that takes the output of CATFlow to detect buses not on time, bicycles not on path, large vehicles during rush hours and pedestrians jaywalking. In addition, SED is used to detect vehicle horns. The data are then collected by the UI where a local authority can compare the data before and after an educational campaign or a traffic calming measure to assess the success of the efforts. The impact of this use case will be measured through interviews with authorities or other third parties to determine if this tool will help them evaluate their campaigns.

2.2.3 GRN Use Case 3 – Traffic Conditions and Anomalous Events

This use case refers to monitoring traffic conditions and detecting anomalous events, such as traffic jams, accidents, congested cars obstructing a junction, very slow vehicles and service vehicles parked on the side or obstructing a carriageway.

The latter event is frequent in Malta's narrow one-way urban streets, often causing cascading effects that extend beyond the immediate area. In general, this output would find application in systems intended to inform drivers near the detected anomaly or to infer possible issues in adjacent areas, thus informing drivers of obstacles ahead. In addition, the detection of anomalous events alerts personnel stationed at traffic management control rooms, who can then interpret the data and take the necessary actions.

The aim of this use case, as described in deliverable D1.2, is to detect anomalous road conditions which may be related directly or indirectly to obstructions. In terms of evaluation metrics, accuracy and detection time are two crucial features of the use case and these can be used to benchmark the system as follows:

- Correct detection of the cause of obstructions on the road 70% of the time, as verified by manual inspection.
- Detection of anomalous events within 2 minutes from their start, as verified by manual inspection.

The implementation of this use case in R1 was designed to address both these goals by flagging an anomaly to traffic personnel in the control room and providing visual insight into the anomaly. The users of the system are intended to be traffic managers who can give directives to authorities to react to a traffic incident.

In R2, the major updates in this use case, following the R1 integration, are improvements to the used infrastructure and associated component deployment, to the AI models used and to the visualisation aspects of the UI.

2.2.4 GRN Use Case 4 – Junction Traffic Trajectory Collection

Junction Traffic Trajectory data collection is focused on the requirement of long-term data analytics that shed light on both the behaviour of road users (e.g., car drivers, motorcyclists, cyclists, pedestrians, etc.) and gathering traffic statistics at road network junctions. This use case is of interest for long-term transport planning and evaluation, particularly in studying active travel modes, such as cycling, walking, and micro-mobility more generally. The innovation envisaged is the construction of a queryable database that can be used to look up historical data on the vehicles and pedestrians in two junctions, with sufficient accuracy to detect anomalous patterns autonomously 50% of the time. The trajectories and data generated from the CATFlow, TAD, SED, and AVCC components are persistently stored on the DFB such that the data can be accessed and processed by the end-user at a time.

In R2, the major updates in this use case, following the R1 integration, are improvements in the used infrastructure and associated component deployment, to the AI models used and to the visualisation aspects of the UI.

2.2.5 MT Use Case 1 – Monitoring of crowded areas

The goal of this use case is to select views of relevant areas for reasons such as exceptional crowd, suspect or unusual crowd movements, etc.

As an organiser of a large public event like the "Christmas Markets" in Trento, it is indeed crucial to have a system in place that can select relevant views of areas based on various factors such as crowd size, suspect behaviour, and unusual crowd movements. The MARVEL framework, integrated with the existing camera infrastructure, can fulfil these requirements and help prevent emergency situations.

By deploying the MARVEL framework, the system continuously analyses the visual data streams from the cameras installed in the squares hosting the "Christmas Markets" (Piazza Fiera and Piazza Duomo). It can accurately estimate the number of visitors in different areas of the event and provide real-time information about their locations. This data is valuable for event organisers to monitor crowd density and make informed decisions regarding crowd management and safety measures.

Furthermore, the MARVEL framework excels at detecting anomalous behaviour within the crowd. It can identify suspicious activities or unusual crowd movements, alerting the control room managed by the local police in real-time. This enables the authorities to take immediate action, if necessary, to prevent any potential emergencies or address emerging issues promptly. By having this early warning system in place, you can ensure that people are alerted, and relevant services are promptly dispatched to handle the situation, especially in overcrowded areas where access to emergency services might be more challenging and slower.

The major improvements in this use case after R1 are related to the AI models used and improvements to the visualisation aspects in SmartViz.

2.2.6 MT Use Case 2 – Detecting criminal/anti-social behaviours

The goal of this use case is to monitor and detect criminal or anti-social behaviours in a proactive approach to ensuring public safety. The integration of visual and audio data streams from existing cameras allows for real-time analysis and timely response to potentially dangerous situations.

When the MARVEL framework identifies a potentially dangerous situation, it triggers an alarm and provides a custom view for further analysis. The system promptly notifies the local police operational centre, providing them with the necessary information to assess the situation and take appropriate action. The local police can dispatch a squad to the location to address the reported incident swiftly.

Furthermore, the system ensures that the visual and audio data streams are saved on the local server of the local police. This enables the police to conduct thorough investigations and gather evidence for any necessary legal proceedings or future reference. The stored data serve as a valuable resource for law enforcement agencies to analyse patterns, identify suspects, or gather additional information related to criminal activities.

By implementing the MARVEL framework in S. Maria Maggiore, the local authorities can enhance their surveillance capabilities and improve public safety. The integration of advanced technology assists law enforcement in their efforts to detect and prevent criminal or antisocial behaviours, ultimately creating a safer environment for residents and visitors alike.

This use case adopts the AAC component to describe what is happening in the audio signal, SED to detect sound events and their temporal location and AVAD to detect anomalies like gatherings, robberies, aggressions, people screaming and drug dealing.

2.2.7 MT Use Case 3 – Monitoring of parking places

In this use case, the MARVEL framework is deployed for audio-visual monitoring of the "Ex Zuffo" Parking Area in Trento. With around 1000 parking places, this parking lot serves as a crucial facility for citizens who utilise public transportation, bike-sharing services, or e-scooters to move around the city centre. The goal is to enhance security and prevent incidents such as car robberies, damages to parked cars, obstructions, and other potential issues through comprehensive analysis of the audio and visual data captured by the existing cameras and microphones installed as part of the MARVEL project.

The MARVEL framework leverages its audio-visual analysis capabilities to monitor and detect various scenarios in real-time. It can analyse the trajectories of cars entering and exiting the parking slots, ensuring that vehicles stay within their designated areas. Any deviations or cars parked outside of the designated slots can be flagged as a potential issue, allowing for timely intervention to address the situation.

Moreover, the system can detect car damages, whether intentional or accidental. By analysing the visual data, the MARVEL framework can identify any signs of vandalism or collisions, promptly notifying the relevant authorities. This enables a swift response to prevent further damage and assist the affected car owners.

The prevention of car robberies is also a crucial aspect of this use case. The MARVEL framework's audio-visual analysis capabilities can detect suspicious activities or individuals near parked cars. It can identify potential theft attempts or unusual behaviour, triggering immediate alerts to the monitoring personnel. This proactive approach helps deter criminals and provides a higher level of security for the parked vehicles.

The major improvements in this use case are related to the AI models used and improvements to the visualisation aspects in SmartViz.

2.2.8 MT Use Case 4 – Analysis of a specific area

By deploying the MARVEL framework in this context, the municipality can gather valuable data and insights to support decision-making and facilitate the development of sustainable mobility and energy plans.

The MARVEL framework's capabilities allow for the counting and tracking of various entities such as persons, cars, buses, taxis, and bikes. By analysing the visual data streams and trajectories, the system can provide accurate and real-time information regarding the movement and distribution of these entities throughout the city. These data can be crucial for understanding traffic patterns, identifying congestion hotspots, and optimising transportation infrastructure and services.

Furthermore, the MARVEL framework can calculate notable events within a specific timeframe. This includes detecting unusual traffic congestion, accidents, or any significant occurrences that may impact the city's operations or require immediate attention. By having access to this information, the municipality can proactively respond to such events, mitigate their impact, and improve overall safety and efficiency within the urban environment.

The framework can provide valuable insights on traffic flows, usage patterns of various transportation modes, and the impact of energy consumption in different areas of the city. This information serves as a solid foundation for the development and monitoring of sustainable mobility plans and energy transition strategies.

The framework will contribute to the effective monitoring and planning of urban systems, ultimately leading to a more sustainable and liveable city environment.

This use case adopts the CATFlow component for counting persons, cars, buses, taxis, and bikes and determining their trajectories, AVAD and SED components to describe what is happening in the audio and video signal and for detecting anomalies like gatherings, robberies, aggressions, people screaming, and drug dealing.

2.2.9 UNS Use Case 1 – Drone experiment

The aim of this use case is to assess the potential use of drones in monitoring large public events held in open spaces. Monitoring and surveillance at large public events can pose challenges due to insufficient infrastructure and the occasionally unpredictable nature of crowds. Fixed street cameras may offer frontal views of the crowd, but they are not able to accurately capture finer details. Overcrowded places are potentially dangerous zones because in the event of an accident, emergency services cannot respond quickly, and people can panic. This motivated us to place visual crowd counting in the core of the UNS1 use case.

Additionally, there are areas or sections in the vicinity of large public events that remain unmonitored due to the lack of fixed cameras and unpredicted presence of people. The presence of people in such areas could be quickly detected using the IFAG MEMS microphones and VAD component while a drone could be sent for inspection, which is supported using defined architecture.

2.2.10 UNS Use Case 2 – Localising audio events in crowds

Monitoring and ensuring safety in rapidly expanding urban city areas as well as public events present complex and demanding tasks, particularly when it comes to promptly responding to anomalous incidents. The conventional approach typically involves the use of fixed cameras for surveillance purposes. While drone cameras and MEMS microphones offer valuable insights, that were analysed within UNS Use Case 1, limitations of video monitoring in scenarios characterised by low visibility or an inadequate number or poor quality of cameras, could be solved using a different approach. In such situations, video monitoring alone is insufficient in providing comprehensive surveillance coverage. As part of this particular use case, our focus shifts to evaluating the potential of implementing microphone array boards for monitoring public events. By harnessing the capabilities of microphone arrays, it becomes possible to detect target sound events and determine the direction of sound propagation. This innovative approach can greatly aid in localising anomalous events, thereby strengthening overall safety measures. By quickly identifying accidents or other forms of abnormal incidents, such a system would significantly enhance the efficiency and effectiveness of event monitoring and response protocols. For such detection, the SELD component is developed within MARVEL and represents a core component within the UNS Use Case 2.

We have defined an experiment consisting of 3 different anomalous sound events (gunshot and gunfire, boom and shatter), combined with a chatter class of events. All data are recorded within staged recording process, using samples from the FSD50K dataset, that are further mixed in order to simulate anomalies in crowds.

Current state-of-the-art sound event localisation and detection components are trained using data simulated and recorded in laboratory conditions, whereas the UNS Use Case 2 included data recording outdoors, which is much closer to the real scenario. Detailed description of data recording process within UNS Use Case 2 is provided in D6.3.

Finally, it should be noted that **the use case "Localising audio events in crowds" replaces the one that had been previously defined for UNS2** ("Audio-visual emotion recognition"). The rationale for this change was based on the ethical challenges identified in the old UNS2 use case and, on the other hand, on the need to develop a novel component for sound event localisation and detection and an accompanying use case. The details of this change can be found in D6.3, Section 2.2.3.

3 R2 Integration Plan and Methodology

During R1 integration activities, a series of challenges were identified that led to the establishment of an integration methodology documented in D5.4. In the framework of R2, many of the same challenges persisted, while a series of new challenges were added, as a result of the R2 objectives (Section 2.1). The overall plan and methodology applied in R1 were found to meet their goals and were therefore re-applied in R2, following modifications to better meet the specific requirements of R2.

In order to address the challenges of R2, a combination of two elements was applied:

(i) An **overarching management mechanism** comprising a **hybrid time plan** and **monitoring mechanisms** of the integration activities. The benefits of an agile approach and incremental development and integration continued to be very relevant, as it was expected that changing, convoluted, and conflicting requirements as well as emergent issues would need to be addressed during the development and integration process. However, it was also recognised that project deadlines would need to be respected and multiple parties from participating project partners would need to be coordinated and remain in sync during the development and integration stages. It was therefore deemed necessary to complement the overall approach with elements of the waterfall method to ensure that a global, phased plan with specific milestones would be followed by all relevant parties to align individual efforts.

(ii) **Methods, practices and tools of agile development** and Continuous Integration / Continuous Delivery (CI/CD). CI/CD allowed for a continuous re-evaluation of specifications to address changing requirements and emerging issues. This approach respects the natural, incremental way of developing complex systems while enabling stakeholders to monitor the implementation progress, give early feedback, and react promptly to potential technical or other obstacles that may arise. Finally, with continuous integration, qualitative, non-functional aspects of the developed platform are considered early on, including interoperability, scalability, accountability, transparency, responsibility, and performance, thus achieving quality assurance in system development iterations and releases.

This section describes the main concepts, methods and tools that were implemented for the development and integration of R2.

3.1 Technical Project Organisation

The organisation of the R2 integration activities needed to deal with the inherent complexity of the need to:

- integrate multiple components (**32**) belonging to **15** partners,
- address **10** use cases belonging to **3** pilots with different system designs,
- access and manage **17** AV data sources from the 3 pilot sites,
- manage the computational infrastructure resources at each pilot distributed across **3** layers (Edge-Fog-Cloud), including **11** infrastructure nodes at the edge, **3** infrastructure nodes at the fog and more than **10** VMs at the cloud,
- perform frequent component deployment, corresponding to managing more than **120** services within the MARVEL Kubernetes cluster,
- carry out thorough testing activities for the aforementioned deployments.

It was therefore deemed essential to establish a very clear plan to organise the overall integration activities and streamline them with other parallel project activities. It was also necessary to apply efficient management procedures for closely monitoring the progress in the implementation of the plan and quickly resolving arising issues. This section refers to the overall organisation for the technical planning and management of the R2 integration activities.

3.1.1 Time Plan

In order to structure the various integration activities and deliver R2 as an operational prototype within the allocated time frame, an initial high-level plan was prepared in the form of a roadmap to set the key intermediate milestones that would need to be reached and the main associated activities.

Subsequently, the high-level plan was further elaborated and a more fine-grained, low-level time plan was established in the form of a Gantt Chart (Figure 1). This Gantt Chart was based on the unit of weeks and provided more details on the necessary actions that would need to be carried out and their positioning in time. It adopted a waterfall approach combined with elements of agile development in the following sense: initial design activities were planned to deliver a concrete preliminary design early on so that they could drive the subsequent development and integration activities. However, in the context of the actual development and integration activities, a more agile approach was followed that allowed a continuous revision of design and integration aspects according to arising issues and feedback from end-users. The agile approach was enabled through other complementary integration tools and methods (e.g., weekly integration meetings, issue tracking system, specification documentation system, etc). Furthermore, in R2, the concept of rolling milestones was introduced which is explained later in this section.

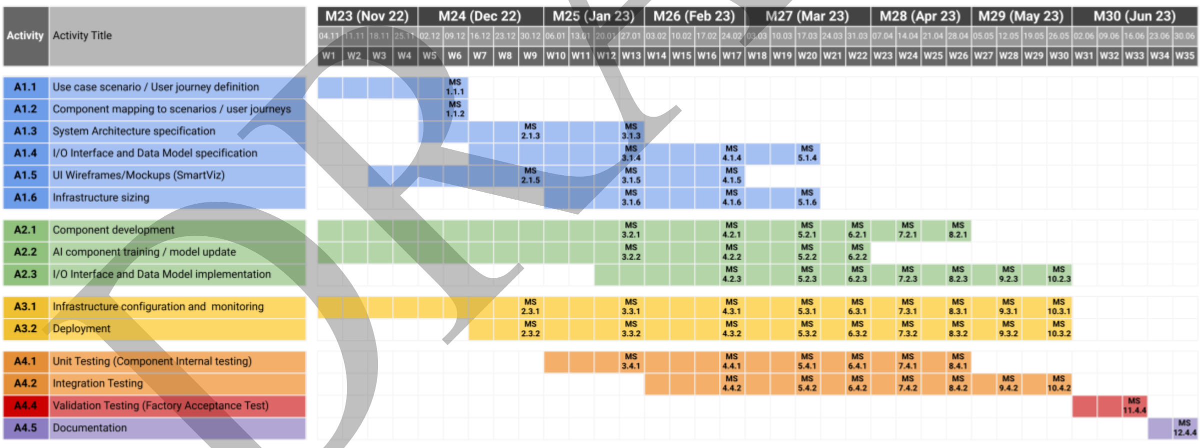


Figure 1: Week-based time plan in the form of a Gantt chart for organising R2 integration activities

The activities in the Gantt Chart were organised in 4 main groups:

1. R2 System Design and preparation

- A1.1 Use case scenario / User journey definition
- A1.2 Component mapping to scenarios / user journeys
- A1.3 System Architecture specification
- A1.4 I/O Interface and Data Model specification
- A1.5 UI Wireframes/Mockups (SmartViz)
- A1.6 Infrastructure sizing

The activities in this group initially focused on achieving a clear definition of the new 5 use cases and associated user journeys from the side of pilot end-users and eliciting the requirements that would drive the R2 system design, following the principles that had already been established in R1. The R2 system design started by mapping components to the defined use cases and Edge-Fog-Cloud layers, also with a view to achieve a representation of all components in R2 and demonstrate the versatility of MARVEL in complying with diverse Edge-Fog-Cloud configurations. The design led to the production of specific system architectures for each addressed use case, as well as API specifications for the communication between components. The approach was complemented with the preparation of UI wireframes and mockups for the MARVEL front-end component (SmartViz). In parallel, an assessment of required infrastructure resources and the intention to use available infrastructure resources efficiently iteratively drove design decisions.

2. Individual component development

- A2.1 Component development
- A2.2 AI component training / model update
- A2.3 I/O Interface and Data Model implementation

This group of activities focused on the development and configuration of individual components. Initially, efforts were made to ensure that all components that were foreseen to be involved in R2, were appropriately configured to be delivered in a containerised form, suitable for deployment in a Kubernetes cluster environment through the MARVdash tool. Further development and optimisation of individual components for achieving the necessary functionalities for R2 took place throughout the entire period of R2 integration activities. Following the specification of APIs and data schemas for communications between components, efforts were also allocated to implementing the associated interfaces and data models at the level of individual components. In parallel, activities included training of the ML models of MARVEL AI components that were applicable in R2, using datasets that were provided by pilot owners.

3. Infrastructure configuration and deployment

- A3.1 Infrastructure configuration and monitoring
- A3.2 Deployment

This group of activities focused on operations associated with the deployment of components to the MARVEL Kubernetes cluster, used for bridging the infrastructure nodes of all pilots and across all layers (Edge, Fog, Cloud) with the support of the MARVdash tool. In the context of these activities, efforts were made to properly configure all infrastructure devices to be seamlessly added to the Kubernetes cluster and host the MARVEL services. In parallel, issues related to data privacy and security concerns were resolved, especially in cases of configuring Edge and Fog host devices. Finally, MARVEL technical partners were active in preparing and updating the container images of their components as well as providing necessary configuration documents to achieve the deployment of the respective services, which was facilitated by MARVdash.

4. Quality Assurance

- A4.1 Unit Testing (Component Internal testing)
- A4.2 Integration Testing
- A4.4 Validation Testing (Factory Acceptance Test)
- A4.5 Documentation

During the development of individual components and implementation of the foreseen APIs, component providers performed rigorous unit testing of the code belonging to each component. Furthermore, a dedicated static code analysis service was established within the common project code repository to assist in this process. Following the implementation of the APIs by individual components, a phase of partial integration testing was initiated, where pairs and groups of components were tested in unison to ensure the foreseen functionality for communications and data exchange between components. These tests were combined with infrastructure stress testing to evaluate the capacity of the infrastructure to host the foreseen components and these tests influenced design decisions. Towards the end of the integration activities, end-to-end integration testing sessions were planned to test the functionality of the entire pipelines and workflows. The R2 integration activities were concluded with the execution of validation tests that comprised formal end-to-end integration tests (Factory Acceptance Tests). Subsequently, R2 integration activities were thoroughly documented in the current D5.6 report.

The time plan was complemented with a series of rolling milestones that were associated with specific activities to ensure the timely delivery of certain results, facilitate cross-dependencies between the various activities and comply with the overall time constraints. These rolling milestones were aligned with the agile methodology that was more closely followed in R2 integration activities, essentially corresponding to iterative updates to the milestones. The integration rolling milestones that were set up and monitored are presented in the following Table 1.

Table 1: R2 Integration Rolling Milestones

MS #	Integration milestone title	Week
MS 1.1.2	Use case scenarios & User journeys defined	W6
MS 1.1.2	Components mapped to scenarios / user journeys	W6
MS X.1.3	System Architecture specification revisions	W9, W13
MS X.1.4	I/O Interface and Data Model specification revisions	W13, W17, W20
MS X.1.5	UI Wireframes/Mockups revisions	W9, W13, W17
MS X.1.6	Infrastructure sizing revisions	W13, W17, W20
MS X.2.1	Component (internal functions) version delivery	W13, 17, W20, W22, W24, W26
MS X.2.2	AI model training version	W13, W17, W20, W22
MS X.2.3	I/O Interface and Data Model implementation version	W17, W20, W22, W24, W26, W28, W30
MS X.3.1	Infrastructure configuration versions	W9, W13, W17, W20, W22, W24, W26, W28, W30
MS X.3.2	Component version deployment	W9, W13, W17, W20, W22, W24, W26, W28, W30
MS X.4.1	Component version Unit Testing execution	W13, W17, W20, W22, W24, W26

MS X.4.2	Integration Testing Iterations	W17, W20, W22, W24, W26, W28, W30
MS 11.4.4	R2 Tested and Qualified	W33
MS 12.4.4	R2 Test Results Documented	W35

3.1.2 Recurring Technical Integration Meetings

In order to deal with the inherent complexity and significant challenges of the integration of the MARVEL framework in R2, the series of recurring meetings that had been originally introduced and established for R1 were resumed. These 'R2 Technical Integration Meetings' focused on the coordination of integration activities from a technical perspective. They were held less frequently than in R1 (on a bi-weekly instead of a weekly basis) because there was a higher demand for E2E integration tests in R2 and therefore testing sessions were organised in the time slot reserved for the integration meetings every two weeks.

The purpose of this series of meetings was to:

- assist in aligning the efforts of technical partners contributing technologies to be integrated into the 2nd Release of the MARVEL Integrated framework (R2),
- track the progress of the integration process,
- identify and resolve issues that would potentially arise.

In these meetings, the participation of technical staff who were directly involved in the development of the MARVEL technological components was promoted.

The goal of these meetings was not to engage in open discussions to resolve peripheral issues in detail, but rather to assist in identifying open issues that would need to be followed up offline or through other focused meetings of the relevant partners.

In total, ten (10) R2 Integration Technical Meetings took place between February and June 2023. The MARVEL Integration Manager organised and moderated this series of meetings.

The initial set of meetings focused on establishing the roadmap and detailed time plan of integration activities. Subsequently, meetings were focused on the definition of the new use cases introduced for R2 and requirement elicitation. In parallel, meetings were referring to design activities, which followed an iterative approach and evolved hand-in-hand with the gradual definition of the new use cases and requirement elicitation. The next set of meetings focused on the specification of APIs and I/O interfaces, data model definition, infrastructure sizing and other implementation specific issues that needed to be addressed, which also affected design decisions (e.g., re-mapping components to other infrastructure nodes based on the stress tests). The focus in the last set of meetings transitioned from issues related to the deployment of specific component service instances to integration testing and issue resolution and bug fixing.

As the R2 Technical Integration meetings progressed, their structure gradually gained greater dependency upon the issue tracking system that was implemented (Section 3.5). Eventually, meetings were entirely organised around the monitoring of the progress of the active issues listed in the issue tracking system, with clear reference to the index of these issues and the offline discourse and progress reported in that system. Meeting discussions at the more mature stages were only focusing on critical issues that were difficult to resolve offline or affected the majority of partners.

3.1.3 Integration Board

The ‘Integration Board’ management body was introduced during R1 integration activities to assist in achieving more efficient management of integration activities. During the R2 integration activities, the ‘Integration Board’ continued to operate. Its purpose was to take a leading role and initiatives in the integration activities and help in swiftly resolving central issues that would arise without needing to involve all technical partners, whenever possible.

The Integration Board was composed of the following members:

- Tassos Kanellos, as a representative of ITML, being the Leader of T5.3 ‘Continuous integration towards MARVEL’s framework realisation’ and Integration Manager.
- Manolis Falelakis, as a representative of INTRA, being the WP5 Leader.
- Dragana Bajovic, as a representative of UNS, being the Scientific and Technical Project Manager (STPM).
- Manos Papoutsakis, as a representative of FORTH, being the Project Coordinator and provider of MARVdash, which is pivotal for the integration and deployment of the MARVEL components.
- Tomás Pariente Lobo, as a representative of ATOS, being the Leader of T5.4 ‘Quantifiable progress against societal, academic and industry validated benchmarks’, as benchmarking is tightly coupled with integration and also being the provider of DatAna, which is pivotal for the integration of multiple components and data exchange.

The Integration Board was chaired by the Integration Manager (ITML representative). The Integration Board members maintained an open communication channel throughout the R2 integration activities in order to prioritise and organise the necessary actions at each stage, after reviewing the achieved progress and evaluating it against objectives that would need to be met. In addition, the Integration Board made organisational decisions and contributed to the overall integration efforts by taking initiatives to propose design solutions and technical configurations that could resolve central issues. Furthermore, the Integration Board performed ad hoc meetings, whenever a high risk was encountered or critical issues arose that could affect multiple integration activities and/or components. These teleconference meetings allowed a more complete understanding of the problems, the identification of mitigation measures and solutions and the prioritisation of actions.

3.2 R2 Design approach

In order to define the function, structure, and form of R2, the R2 integration activities included a track that focused on the design of R2.

The design activities were more intensive at the start of R2 integration and continued until the final stages of integration, gradually decreasing in intensity, in line with the gradual reduction of the initial volatility of the R2 specification.

The design of R2 adopted a dynamic approach, being iteratively revised and optimised during the R2 development and integration activities.

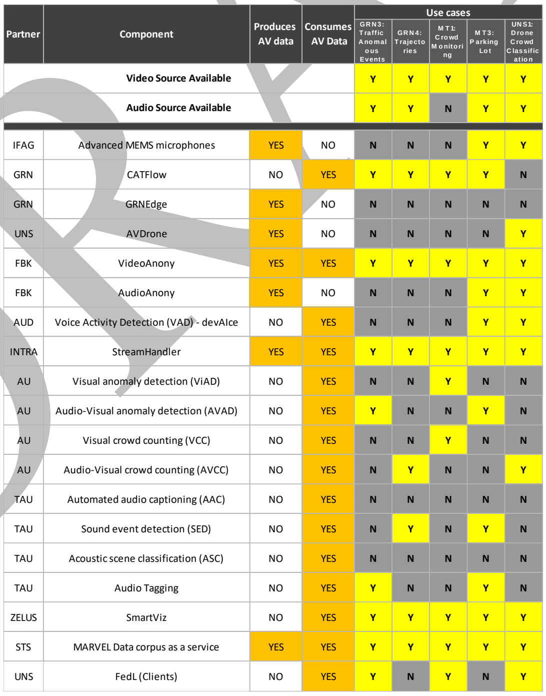
The design activities started with the analysis of the R2 use cases (Section 2.2) and of the available infrastructure.

Regarding the new use cases that were introduced in R2 (GRN1, GRN2, MT2, MT4, UNS2), the analysis led to provisional mappings of components to use cases and infrastructure layers and nodes, in an attempt to meet use case requirements while conforming to the available infrastructure resources. This process gave way to the definition of distinct system architectures for each use case, which was coupled with the identification, specification and documentation of the necessary protocols, I/O interfaces and data models to ensure communication between components. In parallel, the user interfaces and overall user experience was designed, tailored to the needs of the involved end-users in the R2 use cases. At each step, the various aspects of design (e.g., meeting use case requirements, infrastructure sizing, component mapping, interface, and data model specification, UI/UX) were juxtaposed against each other to ensure that all necessary sides of R2 design were being considered and addressed. The process was also influenced by the experience gained from R1 and the relevant technical solutions and conclusions that had been established.

Regarding the use cases that had been addressed by R1, the same design was maintained in R2 to a large extent. The changes that were introduced in these use cases (GRN3, GRN4, MT1, MT3, UNS1) took place to address identified issues, consider changes in the available infrastructure, incorporate some new components and align the approach with the one for the new use cases.

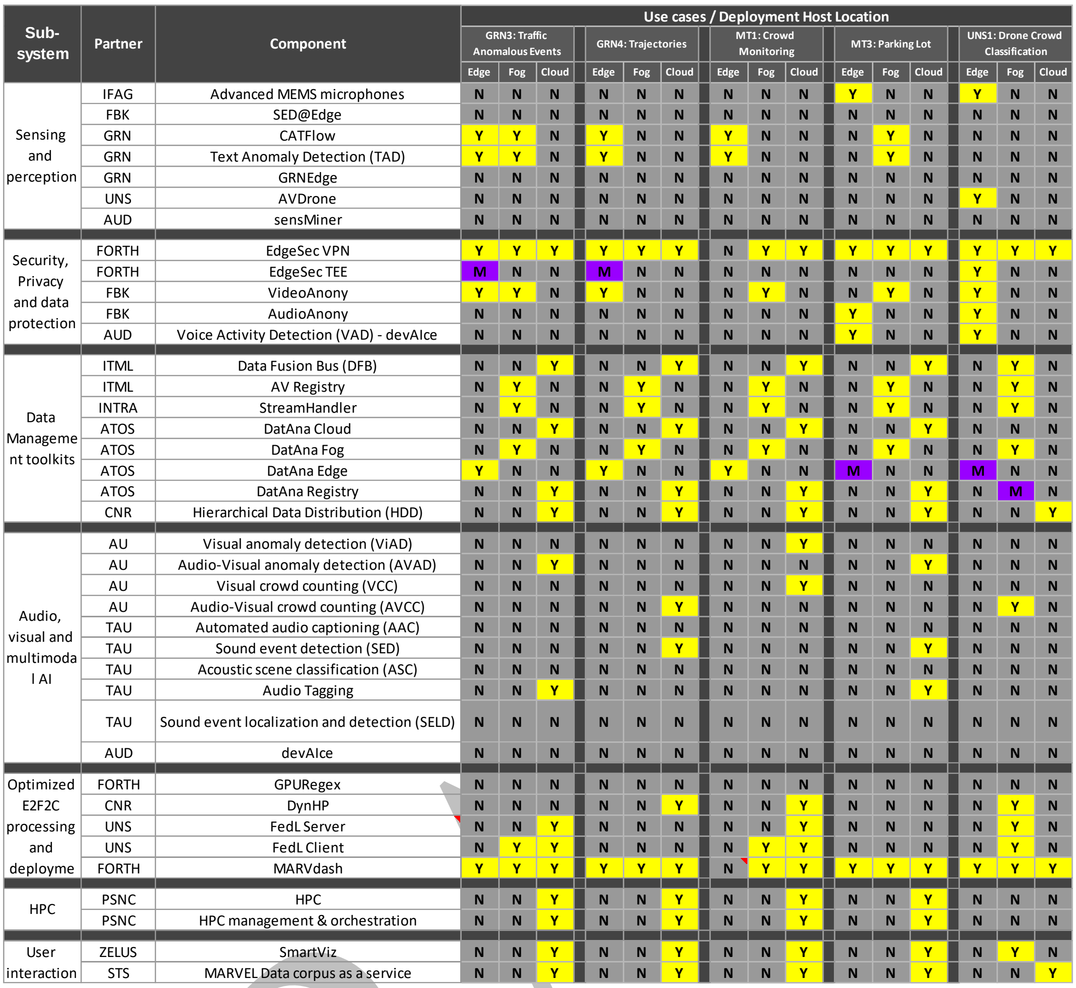
During the design process, the various possibilities were represented using architectural diagrams. Separate diagrams were prepared for the AI inference pipeline of each addressed use case, while a single diagram was prepared for the AI training pipeline, which referred to all use cases. The diagrams were regularly updated, as the R2 design was progressing to provide a common ground for all parties involved in the design process.

The design process was also supported and complemented by a series of central online spreadsheet documents, which were used to document requirements, components, interface, and data model specifications and deployment configurations among other aspects of design in a tabular format. Figure 2 and Figure 3 illustrate some indicative examples.



Partner	Component	Produces AV data	Consumes AV Data	Use cases				
				GRN3: Traffic Anomalous Events	GRN4: Trajectories	MT1: Crowd Monitoring	MT3: Parking Lot	UNS1: Drone Crowd Classification
Video Source Available				Y	Y	Y	Y	Y
Audio Source Available				Y	Y	N	Y	Y
IFAG	Advanced MEMS microphones	YES	NO	N	N	N	Y	Y
GRN	CATFlow	NO	YES	Y	Y	Y	Y	N
GRN	GRNEdge	YES	NO	N	N	N	N	N
UNS	AVDrone	YES	NO	N	N	N	N	Y
FBK	VideoAnony	YES	YES	Y	Y	Y	Y	Y
FBK	AudioAnony	YES	NO	N	N	N	Y	Y
AUD	Voice Activity Detection (VAD) - devAIce	NO	YES	N	N	N	Y	Y
INTRA	StreamHandler	YES	YES	Y	Y	Y	Y	Y
AU	Visual anomaly detection (ViAD)	NO	YES	N	N	Y	N	N
AU	Audio-Visual anomaly detection (AVAD)	NO	YES	Y	N	N	Y	N
AU	Visual crowd counting (VCC)	NO	YES	N	N	Y	N	N
AU	Audio-Visual crowd counting (AVCC)	NO	YES	N	Y	N	N	Y
TAU	Automated audio captioning (AAC)	NO	YES	N	N	N	N	N
TAU	Sound event detection (SED)	NO	YES	N	Y	N	Y	N
TAU	Acoustic scene classification (ASC)	NO	YES	N	N	N	N	N
TAU	Audio Tagging	NO	YES	Y	N	N	Y	N
ZELUS	SmartViz	NO	YES	Y	Y	Y	Y	Y
STS	MARVEL Data corpus as a service	YES	YES	Y	Y	Y	Y	Y
UNS	FedL (Clients)	NO	YES	Y	N	Y	N	Y

Figure 2: MARVEL component requirements regarding AV data



Sub-system	Partner	Component	Use cases / Deployment Host Location														
			GRN3: Traffic Anomalous Events			GRN4: Trajectories			MT1: Crowd Monitoring			MT3: Parking Lot			UNS1: Drone Crowd Classification		
			Edge	Fog	Cloud	Edge	Fog	Cloud	Edge	Fog	Cloud	Edge	Fog	Cloud	Edge	Fog	Cloud
Sensing and perception	IFAG	Advanced MEMS microphones	N	N	N	N	N	N	N	N	N	Y	N	N	Y	N	N
	FBK	SED@Edge	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N
	GRN	CATFlow	Y	Y	N	Y	N	N	Y	N	N	N	Y	N	N	N	N
	GRN	Text Anomaly Detection (TAD)	Y	Y	N	Y	N	N	Y	N	N	N	Y	N	N	N	N
	GRN	GRNEdge	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N
	UNS	AVDrone	N	N	N	N	N	N	N	N	N	N	N	N	Y	N	N
	AUD	sensMiner	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N
Security, Privacy and data protection	FORTH	EdgeSec VPN	Y	Y	Y	Y	Y	Y	N	Y	Y	Y	Y	Y	Y	Y	Y
	FORTH	EdgeSec TEE	M	N	N	M	N	N	N	N	N	N	N	N	Y	N	N
	FBK	VideoAnony	Y	Y	N	Y	N	N	N	Y	N	N	Y	N	Y	N	N
	FBK	AudioAnony	N	N	N	N	N	N	N	N	N	Y	N	N	Y	N	N
	AUD	Voice Activity Detection (VAD) - devAIce	N	N	N	N	N	N	N	N	N	Y	N	N	Y	N	N
Data Management toolkits	ITML	Data Fusion Bus (DFB)	N	N	Y	N	N	Y	N	N	Y	N	N	Y	N	Y	N
	ITML	AV Registry	N	Y	N	N	Y	N	N	Y	N	N	Y	N	N	Y	N
	INTRA	StreamHandler	N	Y	N	N	Y	N	N	Y	N	N	Y	N	N	Y	N
	ATOS	DatAna Cloud	N	N	Y	N	N	Y	N	N	Y	N	N	Y	N	N	N
	ATOS	DatAna Fog	N	Y	N	N	Y	N	N	Y	N	N	Y	N	N	Y	N
	ATOS	DatAna Edge	Y	N	N	Y	N	N	Y	N	N	M	N	N	M	N	N
	ATOS	DatAna Registry	N	N	Y	N	N	Y	N	N	Y	N	N	Y	N	M	N
	CNR	Hierarchical Data Distribution (HDD)	N	N	Y	N	N	Y	N	N	Y	N	N	Y	N	N	Y
Audio, visual and multimodal AI	AU	Visual anomaly detection (ViAD)	N	N	N	N	N	N	N	N	Y	N	N	N	N	N	N
	AU	Audio-Visual anomaly detection (AVAD)	N	N	Y	N	N	N	N	N	N	N	N	Y	N	N	N
	AU	Visual crowd counting (VCC)	N	N	N	N	N	N	N	N	Y	N	N	N	N	N	N
	AU	Audio-Visual crowd counting (AVCC)	N	N	N	N	N	Y	N	N	N	N	N	N	N	Y	N
	TAU	Automated audio captioning (AAC)	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N
	TAU	Sound event detection (SED)	N	N	N	N	N	Y	N	N	N	N	N	Y	N	N	N
	TAU	Acoustic scene classification (ASC)	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N
	TAU	Audio Tagging	N	N	Y	N	N	N	N	N	N	N	N	Y	N	N	N
	TAU	Sound event localization and detection (SELD)	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N
	AUD	devAIce	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N
Optimized E2F2C processing and deployme	FORTH	GPURegex	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N
	CNR	DynHP	N	N	N	N	N	Y	N	N	Y	N	N	N	N	Y	N
	UNS	FedL Server	N	N	Y	N	N	N	N	N	Y	N	N	N	N	Y	N
	UNS	FedL Client	N	Y	Y	N	N	N	N	Y	Y	N	N	N	N	Y	N
	FORTH	MARVdash	Y	Y	Y	Y	Y	Y	N	Y	Y	Y	Y	Y	Y	Y	Y
HPC	PSNC	HPC	N	N	Y	N	N	Y	N	N	Y	N	N	Y	N	N	N
	PSNC	HPC management & orchestration	N	N	Y	N	N	Y	N	N	Y	N	N	Y	N	N	N
User interaction	ZELUS	SmartViz	N	N	Y	N	N	Y	N	N	Y	N	N	Y	N	Y	N
	STS	MARVEL Data corpus as a service	N	N	Y	N	N	Y	N	N	Y	N	N	Y	N	N	Y

Figure 3: MARVEL component mapping to use cases and deployment location at infrastructure layers

3.3 Infrastructure sizing

In order to achieve a functional operation of the MARVEL system, it was required to allocate the necessary infrastructure resources that could support the foreseen system design and guarantee an uninterruptable system operation. For the most part, the infrastructure resources refer to the devices that could be attached to the MARVEL Kubernetes cluster and host the services that were to be deployed. It was therefore necessary to perform certain estimations regarding the infrastructure resources in terms of computational power (CPU and GPU cores), memory capacity (RAM and VRAM size), hard disk capacity and performance, and network performance.

A dual process was followed to perform the estimations for infrastructure resources. On one hand, the MARVEL partners attempted to estimate the requirements of each component that was foreseen to be deployed in the context of R2. On the other hand, the MARVEL Consortium carefully examined and documented the specifications of the infrastructure devices that could potentially be made available. During the initial stages of the system design, both aspects were taken into consideration in order to design a system that could (a) meet the end-user requirements derived from the specified use cases and (b) fit the components under consideration to the infrastructure resources that could be allocated. Therefore, the MARVEL Consortium attempted to select and map the available components to the foreseen use cases, to the 3 EFC layers and to specific available devices, while also considering the payload as a variable, e.g., how many data sources would need to be processed in each case. This was an iterative process that was also continued after the initial design had been concluded, as more information became available (e.g., benchmark test results, possibilities to increase the infrastructure resources). The objective was to balance conflicting requirements making efficient compromises that would allow an effective demonstration of the capabilities of MARVEL in addressing the use cases as a unified system.

Information on the estimation of required infrastructure resources for each component was collected using a central tabular document in the format that can be seen in **Table 2**.

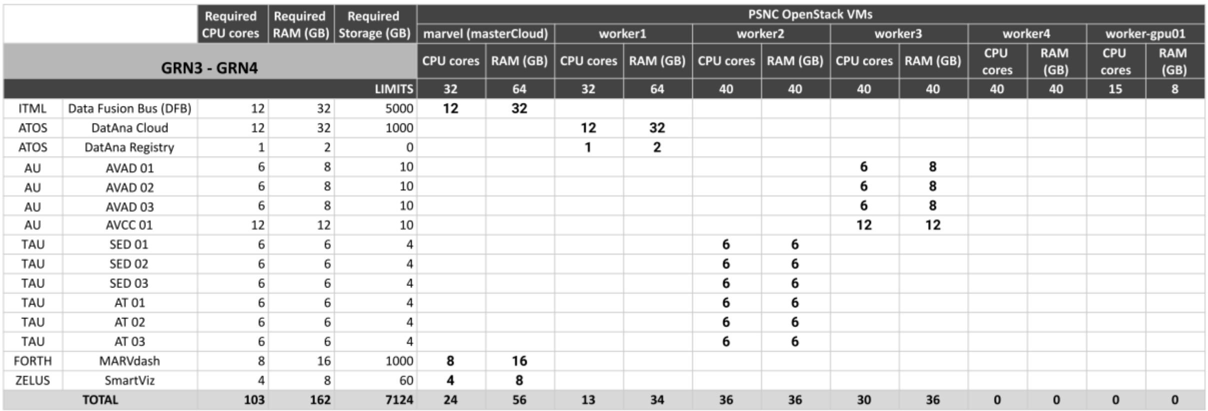
Table 2: Collected component requirement estimations for infrastructure sizing

Component name	Operation Mode	Compatible host device type	Required CPU cores	Required RAM (GB)	GPU required	Required VRAM (GB)	Required Storage (GB)	Nominal payload reference	Requirement Scaling
	• Inference • Training	• Nvidia Jetson • Windows OS PC • Linux OS PC • Fog Server • HPC Infrastructure	Number of CPU cores	Amount of RAM in GB	• Yes • No	Amount of VRAM in GB (if applicable)	Amount of storage in GB	The assumptions used to make the estimations (e.g., process 1 FHD video source)	Any indications for the type of scaling of requirements according to payload (e.g., linear or speedup metrics).

The collected information referred to single component instances. In cases where additional component service instances would need to be deployed on the same host device (e.g., when processing multiple input sources), the aggregate requirements would be estimated, taking into consideration the nominal payload reference that was used to produce the estimations for each instance and any information related to how the service requirements were expected to scale depending on the payload (e.g., speedup metrics).

In the case of the Cloud HPC infrastructure, where multiple components and their instances were foreseen to be deployed, an aggregation of requirements per use case was performed to ensure that the necessary resources could be allocated for the VMs that would host the MARVEL Kubernetes cluster and deployed services.

In R2, it was also attempted to prepare a plan for allocating specific MARVEL services to specific VMs on the Cloud HPC to ensure that services would have access to the necessary resources. An example of such a plan that considers the GRN3 and GRN4 AI components can be seen in Figure 4. However, eventually, it was found that the inherent automated load balancing mechanisms of the Kubernetes framework can successfully manage the service allocation to available VMs without any issues arising and therefore the hard-coded allocation plans were maintained as backups.



		Required CPU cores	Required RAM (GB)	Required Storage (GB)	PSNC OpenStack VMs											
					marvel (masterCloud)		worker1		worker2		worker3		worker4		worker-gpu01	
GRN3 - GRN4					CPU cores	RAM (GB)	CPU cores	RAM (GB)	CPU cores	RAM (GB)	CPU cores	RAM (GB)	CPU cores	RAM (GB)	CPU cores	RAM (GB)
				LIMITS	32	64	32	64	40	40	40	40	40	40	15	8
ITML	Data Fusion Bus (DFB)	12	32	5000	12	32										
ATOS	DatAna Cloud	12	32	1000			12	32								
ATOS	DatAna Registry	1	2	0			1	2								
AU	AVAD 01	6	8	10							6	8				
AU	AVAD 02	6	8	10							6	8				
AU	AVAD 03	6	8	10							6	8				
AU	AVCC 01	12	12	10							12	12				
TAU	SED 01	6	6	4					6	6						
TAU	SED 02	6	6	4					6	6						
TAU	SED 03	6	6	4					6	6						
TAU	AT 01	6	6	4					6	6						
TAU	AT 02	6	6	4					6	6						
TAU	AT 03	6	6	4					6	6						
FORTH	MARVdash	8	16	1000	8	16										
ZELUS	SmartViz	4	8	60	4	8										
TOTAL		103	162	7124	24	56	13	34	36	36	30	36	0	0	0	0

Figure 4: Service allocation plan to specific PSNC HPC VMs to ensure resource availability

3.4 Source version control system

A GitLab[6] repository had been set up in the context of R1 development and integration activities at a VM hosted at the PSNC HPC infrastructure that acted as the main MARVEL source version control system. This repository continued to be used during R2 development.

Individual projects had been set up within the repository for each MARVEL component and user accounts for all participating partner members. The repository was available to all partners wishing to use it for version control of the code of their component that was under development. In addition, a SonarQube[7] service was deployed and integrated with the GitLab repository. The service was made available to all partners for static code analysis and automated unit testing in the context of Quality Assurance (see Section 3.8).

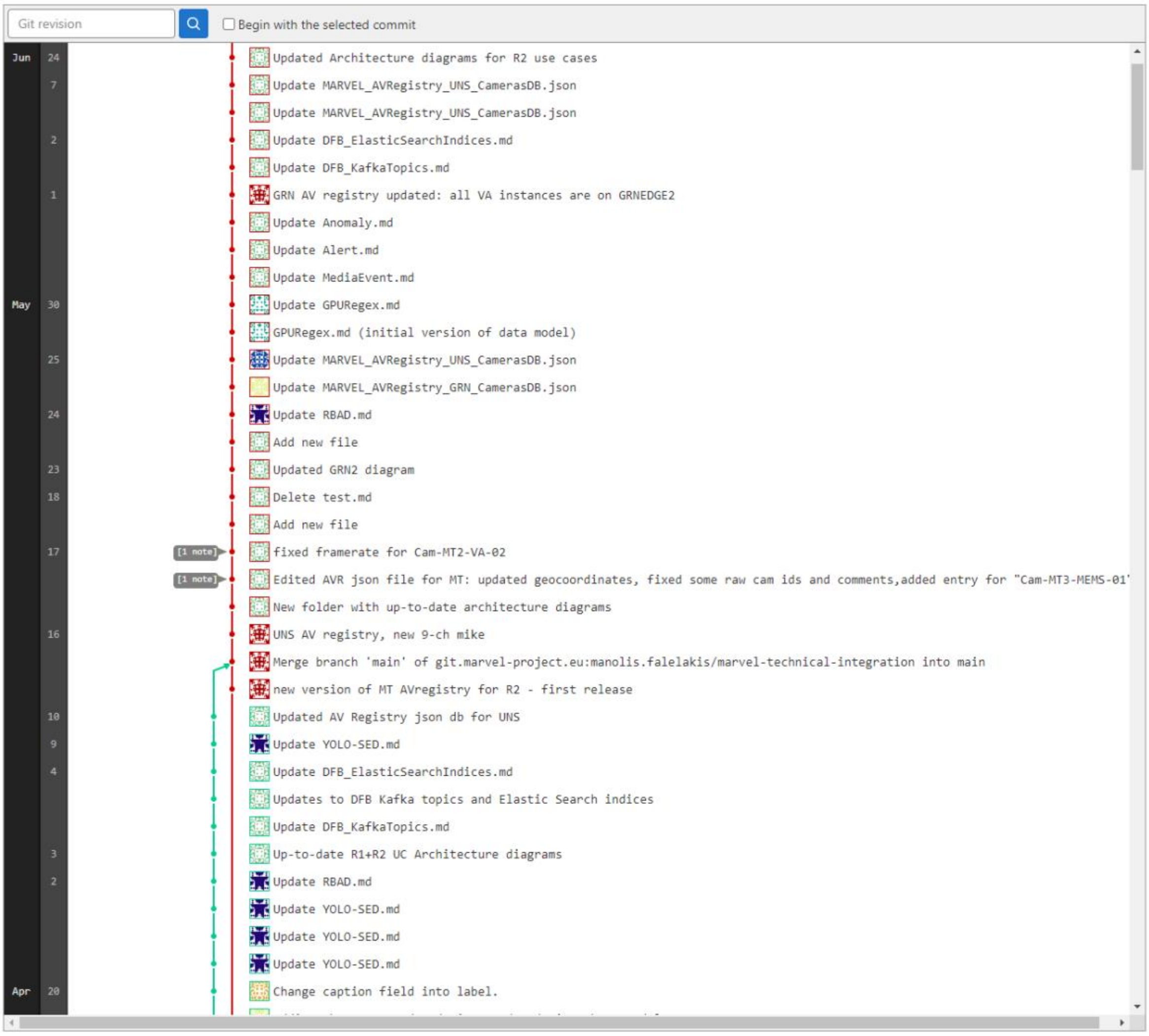
A project that was used exclusively for integration purposes was also created within the GitLab repository. This project hosted:

- The central issue tracking system for monitoring and resolving integration issues (see Section 3.5).
- A repository of open-source code that was useful to multiple parties using version control.
- Documentation of the integration specifications (API and data model specifications) using version control (see Section 3.6)
- Up-to-date data stores and operational information that was required by individual components (e.g., Camera JSON objects for each pilot for the AV Registry component) using version control.

By the end of R2 integration activities, more than 300 commits had been registered in the main branch of the central project (Figure 5).

[6] https://gitlab.com/

[7] https://www.sonarqube.org/



Git revision

□ Begin with the selected commit

```
Jun 24            Updated Architecture diagrams for R2 use cases
     7            Update MARVEL_AVRegistry_UNS_CamerasDB.json
                  Update MARVEL_AVRegistry_UNS_CamerasDB.json
     2            Update DFB_ElasticSearchIndices.md
                  Update DFB_KafkaTopics.md
     1            GRN AV registry updated: all VA instances are on GRNEDGE2
                  Update Anomaly.md
                  Update Alert.md
                  Update MediaEvent.md
May 30            Update GPURegex.md
                  GPURegex.md (initial version of data model)
    25            Update MARVEL_AVRegistry_UNS_CamerasDB.json
                  Update MARVEL_AVRegistry_GRN_CamerasDB.json
    24            Update RBAD.md
                  Add new file
    23            Updated GRN2 diagram
    18            Delete test.md
                  Add new file
    17  [1 note]  fixed framerate for Cam-MT2-VA-02
        [1 note]  Edited AVR json file for MT: updated geocoordinates, fixed some raw cam ids and comments,added entry for "Cam-MT3-MEMS-01'
                  New folder with up-to-date architecture diagrams
    16            UNS AV registry, new 9-ch mike
                  Merge branch 'main' of git.marvel-project.eu:manolis.falelakis/marvel-technical-integration into main
                  new version of MT AVregistry for R2 - first release
    10            Updated AV Registry json db for UNS
     9            Update YOLO-SED.md
     4            Update DFB_ElasticSearchIndices.md
                  Updates to DFB Kafka topics and Elastic Search indices
                  Update DFB_KafkaTopics.md
     3            Up-to-date R1+R2 UC Architecture diagrams
     2            Update RBAD.md
                  Update YOLO-SED.md
                  Update YOLO-SED.md
                  Update YOLO-SED.md
Apr 20            Change caption field into label.
```

Figure 5: Commits in the main branch of the repository used for integration of the MARVEL framework

3.5 Issue Tracking system

The integration activities were supported from the early stages of R1 by a dedicated Issue Tracking System (ITS) that was established. This ITS continued to be used more extensively during the R2 integration. The ITS was hosted within the MARVEL GitLab repository (see Section 3.4 and was accessible by all partners.

The purpose of the ITS was to:

- **Document open issues that required actions from single or multiple parties**. The topics of issues varied considerably and included aspects of design, documentation, deployment, test preparation and bug fixing among others. Each issue was assigned a title and an index that could be cross-referenced among the listed issues. Each issue also contained a text description, formatted with the GitLab Flavored Markdown (GLFM) syntax (Figure 6).
- **Assign issues with actionable items to specific partners/persons.** Each issue was assigned to a single or multiple parties and required specific actions to be taken. Results were required to be reported under the original issue description. In addition, relevant parties were also associated with issues using a monitoring status so that they could be aware of the progress and actions taken for resolving it and provide feedback when

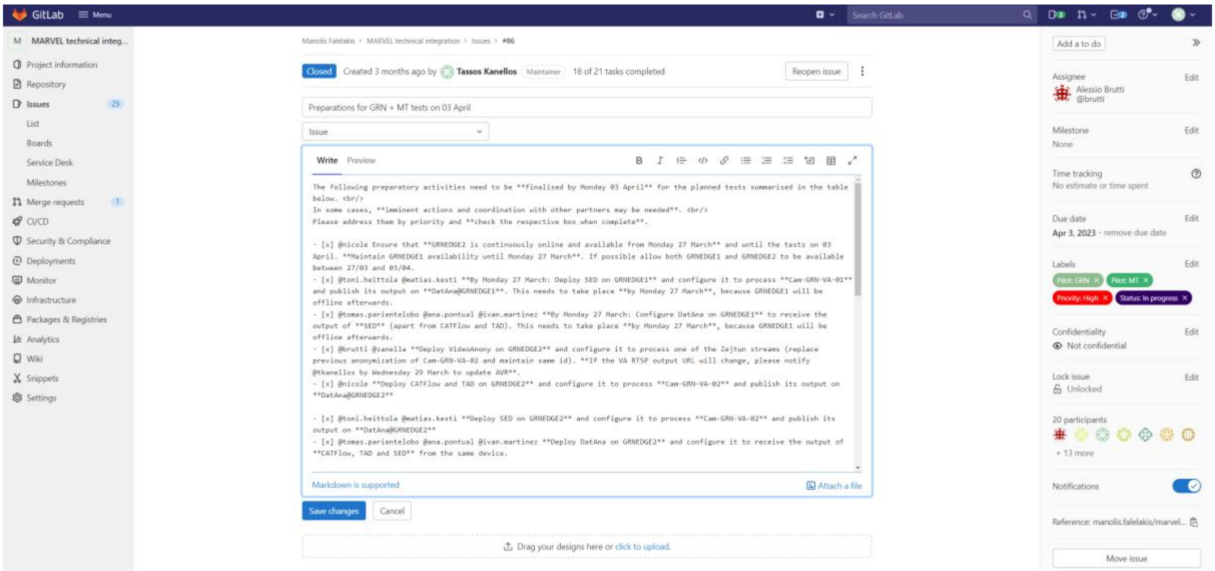
necessary. The overall procedure for managing issues through the ITS contributed to increased accountability, transparency, and responsibility, in line with the Task T5.3 objectives.

- **Serve as a communication channel**, in cases of issues that required actions by multiple parties or collaborative efforts. The GitLab ITS allows the maintenance of discussion threads with comments and responses that are attached sequentially to original issue descriptions (Figure 7). This facilitated asynchronous communication and exchange of information related to specific issues at a central shared location. This process ensured continuity and the formation of a common, persistent knowledge base, where information can be tracked and re-visited, thus presenting clear advantages over other communication methods (e.g., disparate email exchanges).
- **Organise and prioritise issues, including deadlines and association with milestones.** The GitLab ITS labelling feature was used to annotate issues and provide the means for categorising, filtering, and quickly retrieving the necessary information. The labels that were implemented referred to the pilots related to the issue, the issue priority (low, medium, high), and the issue status (backlog, open, closed). Each issue was also given a deadline for carrying out the necessary actions and was associated with a milestone corresponding to the weeks allocated for integration activities according to the overall time plan.
- **Monitor and manage the progress of integration activities**. The ITS allowed the close monitoring of the progress in multiple, parallel issues and a more efficient management of integration activities through the prioritisation of relevant issues. GitLab allows a comprehensive view of all issues in a list (Figure 8) that is helpful for reviewing issues in relation to the associated deadline and milestone. It also allows a 'board view' (Figure 9), typically associated with Kanban agile methods, which was helpful in the continuous re-evaluation of objectives and prioritisation of issues in the context of the MARVEL integration activities.
- **Maintain a backlog of unresolved items for long-term planning**. The GitLab ITS was also useful for documenting issues for future reference, in the form of a backlog. This process allowed to quickly generate new issues in the form of notes with minimal information, ensuring the documentation and maintenance of items that would be necessary to be addressed at a later date. When the overall integration conditions and priorities were suitable, the issue would receive a more detailed description and would be transferred from the backlog to a status of ongoing tasks.
- **Link issues with parts of code, API, and data model specifications, hosted in the GitLab repository**. Since the MARVEL GitLab repository hosting the ITS also hosted version-controlled code as well as API and data model specifications, it was particularly easy to cross-reference such items within issues.

The ITS was maintained and managed by the Integration Manager, opening new issues and regularly updating existing ones.

By the end of the R2 integration activities, more than 120 issues had been registered. Several issues led to discussion threads with more than 10 comments and responses exchanged between relevant parties.

The ITS was also closely coupled with the R2 Technical Integration Meetings (Section 3.1.2), whose main purpose after the initial integration stages was to collectively review the status of open issues registered in the ITS to organise the next steps.



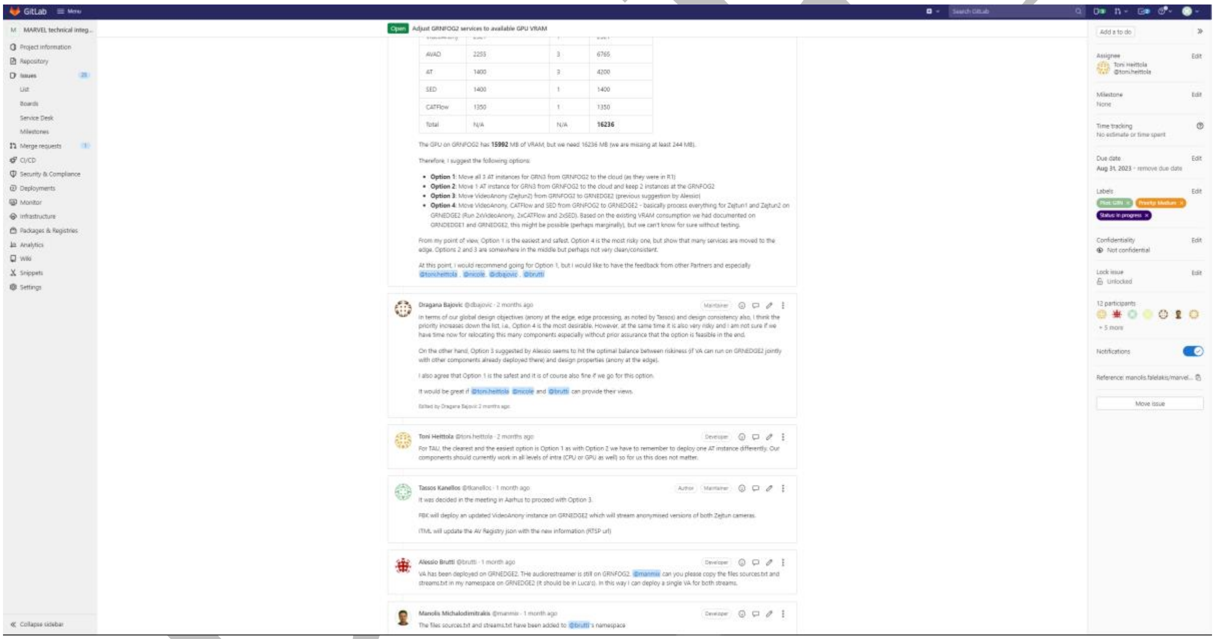
Closed Created 3 months ago by Tassos Kanellos Maintainer 18 of 21 tasks completed

Preparations for GRN + MT tests on 03 April

Issue

```
The following preparatory activities need to be **finalised by Monday 03 April** for the planned tests summarised in the table
below. <br/>
In some cases, **imminent actions and coordination with other partners may be needed**. <br/>
Please address them by priority and **check the respective box when complete**.

- [x] @nicole Ensure that **GRNEDGE2 is continuously online and available from Monday 27 March** and until the tests on 03
April. **Maintain GRNEDGE1 availability until Monday 27 March**. If possible allow both GRNEDGE1 and GRNEDGE2 to be available
between 27/03 and 03/04.
- [x] @toni.heittola @matias.kesti **By Monday 27 March: Deploy SED on GRNEDGE1** and configure it to process **Cam-GRN-VA-01**
and publish its output on **DatAna@GRNEDGE1**. This needs to take place **by Monday 27 March**, because GRNEDGE1 will be
offline afterwards.
- [x] @tomas.parientelobo @ana.pontual @ivan.martinez **By Monday 27 March: Configure DatAna on GRNEDGE1** to receive the
output of **SED** (apart from CATFlow and TAD). This needs to take place **by Monday 27 March**, because GRNEDGE1 will be
offline afterwards.
- [x] @brutti @zanella **Deploy VideoAnony on GRNEDGE2** and configure it to process one of the Zejtun streams (replace
previous anonymization of Cam-GRN-VA-02 and maintain same id). **If the VA RTSP output URL will change, please notify
@tkanellos by Wednesday 29 March to update AVR**.
- [x] @nicole **Deploy CATFlow and TAD on GRNEDGE2** and configure it to process **Cam-GRN-VA-02** and publish its output on
**DatAna@GRNEDGE2**

- [x] @toni.heittola @matias.kesti **Deploy SED on GRNEDGE2** and configure it to process **Cam-GRN-VA-02** and publish its
output on **DatAna@GRNEDGE2**
- [x] @tomas.parientelobo @ana.pontual @ivan.martinez **Deploy DatAna on GRNEDGE2** and configure it to receive the output of
**CATFlow, TAD and SED** from the same device.
```

Figure 6: Generation of a new issue in the GitLab Issue Tracking System using the GitLab Flavoured Markdown (GLFM) syntax and specifying associated issue metadata



Open Adjust GRNFOG2 services to available GPU VRAM

[illegible]	[illegible]	[illegible]	[illegible]
AVAD	2255	3	6765
AT	1400	3	4200
SED	1400	1	1400
CATFlow	1350	1	1350
Total	N/A	N/A	**16236**

The GPU on GRNFOG2 has **15992** MB of VRAM, but we need 16236 MB (we are missing at least 244 MB).

Therefore, I suggest the following options:

- **Option 1**: Move all 3 AT instances for GRN3 from GRNFOG2 to the cloud (as they were in R1)
- **Option 2**: Move 1 AT instance for GRN3 from GRNFOG2 to the cloud and keep 2 instances at the GRNFOG2
- **Option 3**: Move VideoAnony (Zejtun2) from GRNFOG2 to GRNEDGE2 (previous suggestion by Alessio)
- **Option 4**: Move VideoAnony, CATFlow and SED from GRNFOG2 to GRNEDGE2 - basically process everything for Zejtun1 and Zejtun2 on GRNEDGE2 (Run 2xVideoAnony, 2xCATFlow and 2xSED). Based on the existing VRAM consumption we had documented on GRNDEDGE1 and GRNEDGE2, this might be possible (perhaps marginally), but we can't know for sure without testing.

From my point of view, Option 1 is the easiest and safest. Option 4 is the most risky one, but show that many services are moved to the edge. Options 2 and 3 are somewhere in the middle but perhaps not very clean/consistent.

At this point, I would recommend going for Option 1, but I would like to have the feedback from other Partners and especially @toni.heittola , @nicole , @dbajovic , @brutti

Dragana Bajovic @dbajovic · 2 months ago Maintainer

In terms of our global design objectives (anony at the edge, edge processing, as noted by Tassos) and design consistency also, I think the priority increases down the list, i.e., Option 4 is the most desirable. However, at the same time it is also very risky and I am not sure if we have time now for relocating this many components especially without prior assurance that the option is feasible in the end.

On the other hand, Option 3 suggested by Alessio seems to hit the optimal balance between riskiness (if VA can run on GRNEDGE2 jointly with other components already deployed there) and design properties (anony at the edge).

I also agree that Option 1 is the safest and it is of course also fine if we go for this option.

It would be great if @toni.heittola @nicole and @brutti can provide their views.

Edited by Dragana Bajovic 2 months ago

Toni Heittola @toni.heittola · 2 months ago Developer

For TAU, the clearest and the easiest option is Option 1 as with Option 2 we have to remember to deploy one AT instance differently. Our components should currently work in all levels of infra (CPU or GPU as well) so for us this does not matter.

Tassos Kanellos @tkanellos · 1 month ago Author Maintainer

It was decided in the meeting in Aarhus to proceed with Option 3.

FBK will deploy an updated VideoAnony instance on GRNEDGE2 which will stream anonymised versions of both Zejtun cameras.

ITML will update the AV Registry json with the new information (RTSP url)

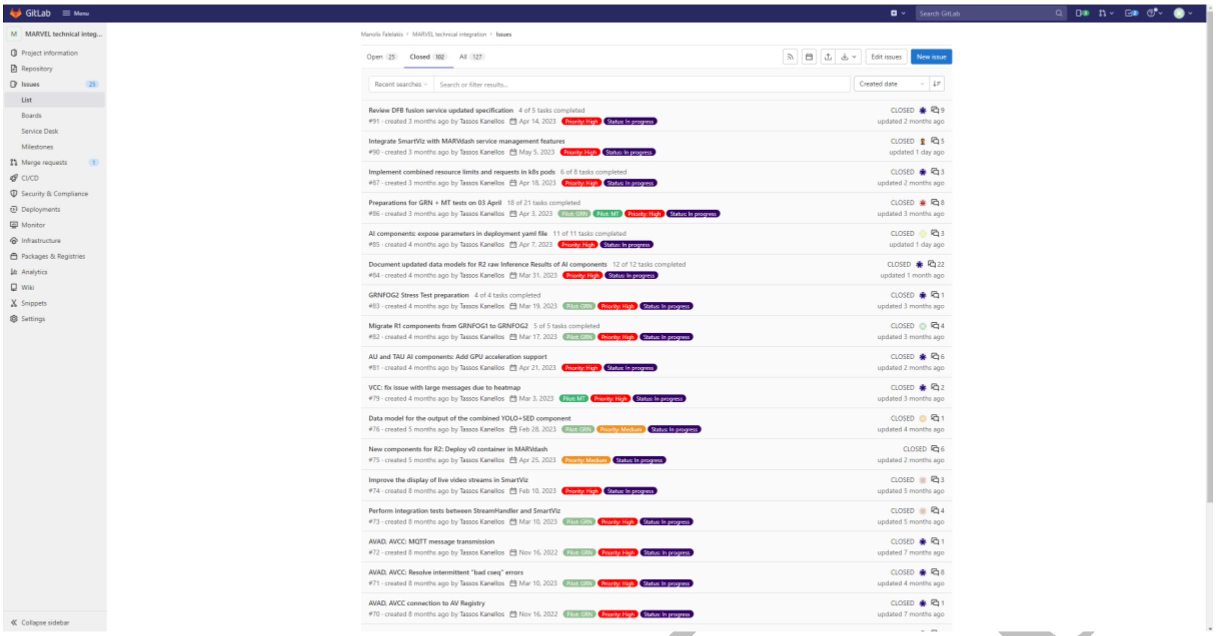
Alessio Brutti @brutti · 1 month ago Developer

VA has been deployed on GRNEDGE2. THe audiorestreamer is still on GRNFOG2. @manmix can you please copy the files sources.txt and streams.txt in my namespace on GRNEDGE2 (it should be in Luca's). In this way I can deploy a single VA for both streams.

Manolis Michalodimitrakis @manmix · 1 month ago Developer

The files sources.txt and streams.txt have been added to @brutti 's namespace

Figure 7: An active discussion thread opened under a specific issue



Review DFB fusion service updated specification 4 of 5 tasks completed
#91 - created 3 months ago by Tassos Kanellos Apr 14, 2023 Priority: High Status: In progress CLOSED 9 updated 2 months ago

Integrate SmartViz with MARVdash service management features
#90 - created 3 months ago by Tassos Kanellos May 5, 2023 Priority: High Status: In progress CLOSED 5 updated 1 day ago

Implement combined resource limits and requests in k8s pods 6 of 8 tasks completed
#87 - created 3 months ago by Tassos Kanellos Apr 18, 2023 Priority: High Status: In progress CLOSED 3 updated 2 months ago

Preparations for GRN + MT tests on 03 April 18 of 21 tasks completed
#86 - created 3 months ago by Tassos Kanellos Apr 3, 2023 Pilot: GRN Pilot: MT Priority: High Status: In progress CLOSED 8 updated 3 months ago

AI components: expose parameters in deployment yaml file 11 of 11 tasks completed
#85 - created 4 months ago by Tassos Kanellos Apr 7, 2023 Priority: High Status: In progress CLOSED 3 updated 1 day ago

Document updated data models for R2 raw Inference Results of AI components 12 of 12 tasks completed
#84 - created 4 months ago by Tassos Kanellos Mar 31, 2023 Priority: High Status: In progress CLOSED 22 updated 1 month ago

GRNFOG2 Stress Test preparation 4 of 4 tasks completed
#83 - created 4 months ago by Tassos Kanellos Mar 19, 2023 Pilot: GRN Priority: High Status: In progress CLOSED 1 updated 3 months ago

Migrate R1 components from GRNFOG1 to GRNFOG2 5 of 5 tasks completed
#82 - created 4 months ago by Tassos Kanellos Mar 17, 2023 Pilot: GRN Priority: High Status: In progress CLOSED 4 updated 3 months ago

AU and TAU AI components: Add GPU acceleration support
#81 - created 4 months ago by Tassos Kanellos Apr 21, 2023 Priority: High Status: In progress CLOSED 6 updated 2 months ago

VCC: fix issue with large messages due to heatmap
#79 - created 4 months ago by Tassos Kanellos Mar 3, 2023 Pilot: MT Priority: High Status: In progress CLOSED 2 updated 3 months ago

Data model for the output of the combined YOLO+SED component
#76 - created 5 months ago by Tassos Kanellos Feb 28, 2023 Pilot: GRN Priority: Medium Status: In progress CLOSED 1 updated 4 months ago

New components for R2: Deploy v0 container in MARVdash
#75 - created 5 months ago by Tassos Kanellos Apr 25, 2023 Priority: Medium Status: In progress CLOSED 6 updated 2 months ago

Improve the display of live video streams in SmartViz
#74 - created 8 months ago by Tassos Kanellos Feb 10, 2023 Priority: High Status: In progress CLOSED 3 updated 5 months ago

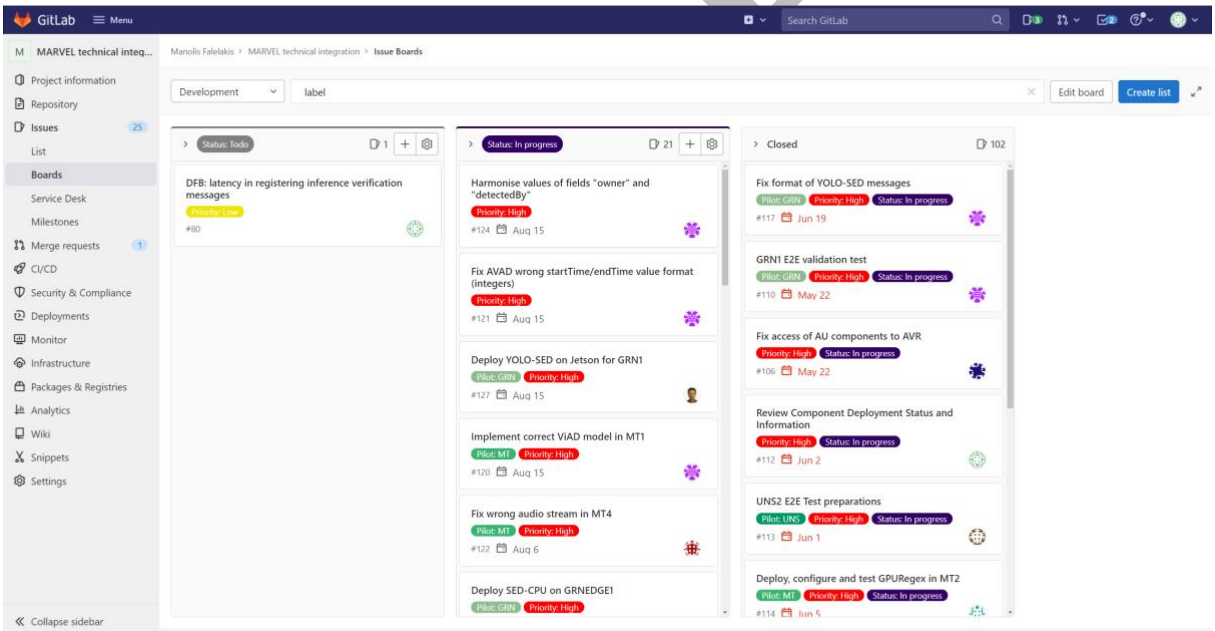
Perform integration tests between StreamHandler and SmartViz
#73 - created 8 months ago by Tassos Kanellos Mar 10, 2023 Pilot: GRN Priority: High Status: In progress CLOSED 4 updated 5 months ago

AVAD, AVCC: MQTT message transmission
#72 - created 8 months ago by Tassos Kanellos Nov 16, 2022 Pilot: GRN Priority: High Status: In progress CLOSED 1 updated 7 months ago

AVAD, AVCC: Resolve intermittent "bad cseq" errors
#71 - created 8 months ago by Tassos Kanellos Mar 10, 2023 Pilot: GRN Priority: High Status: In progress CLOSED 8 updated 4 months ago

AVAD, AVCC connection to AV Registry
#70 - created 8 months ago by Tassos Kanellos Nov 16, 2022 Pilot: GRN Priority: High Status: In progress CLOSED 1 updated 7 months ago

Figure 8: List view of open issues in the MARVEL Integration Issue Tracking System



Status: Todo 1

DFB: latency in registering inference verification messages
Priority: Low
#80

Status: In progress 21

Harmonise values of fields "owner" and "detectedBy"
Priority: High
#124 Aug 15

Fix AVAD wrong startTime/endTime value format (integers)
Priority: High
#121 Aug 15

Deploy YOLO-SED on Jetson for GRN1
Pilot: GRN Priority: High
#127 Aug 15

Implement correct ViAD model in MT1
Pilot: MT Priority: High
#120 Aug 15

Fix wrong audio stream in MT4
Pilot: MT Priority: High
#122 Aug 6

Deploy SED-CPU on GRNEDGE1
Pilot: GRN Priority: High

Closed 102

Fix format of YOLO-SED messages
Pilot: GRN Priority: High Status: In progress
#117 Jun 19

GRN1 E2E validation test
Pilot: GRN Priority: High Status: In progress
#110 May 22

Fix access of AU components to AVR
Priority: High Status: In progress
#106 May 22

Review Component Deployment Status and Information
Priority: High Status: In progress
#112 Jun 2

UNS2 E2E Test preparations
Pilot: UNS Priority: High Status: In progress
#113 Jun 1

Deploy, configure and test GPURegex in MT2
Pilot: MT Priority: High Status: In progress
#114 Jun 5

Figure 9: Board view (Kanban) of open issues in the MARVEL Integration Issue Tracking System

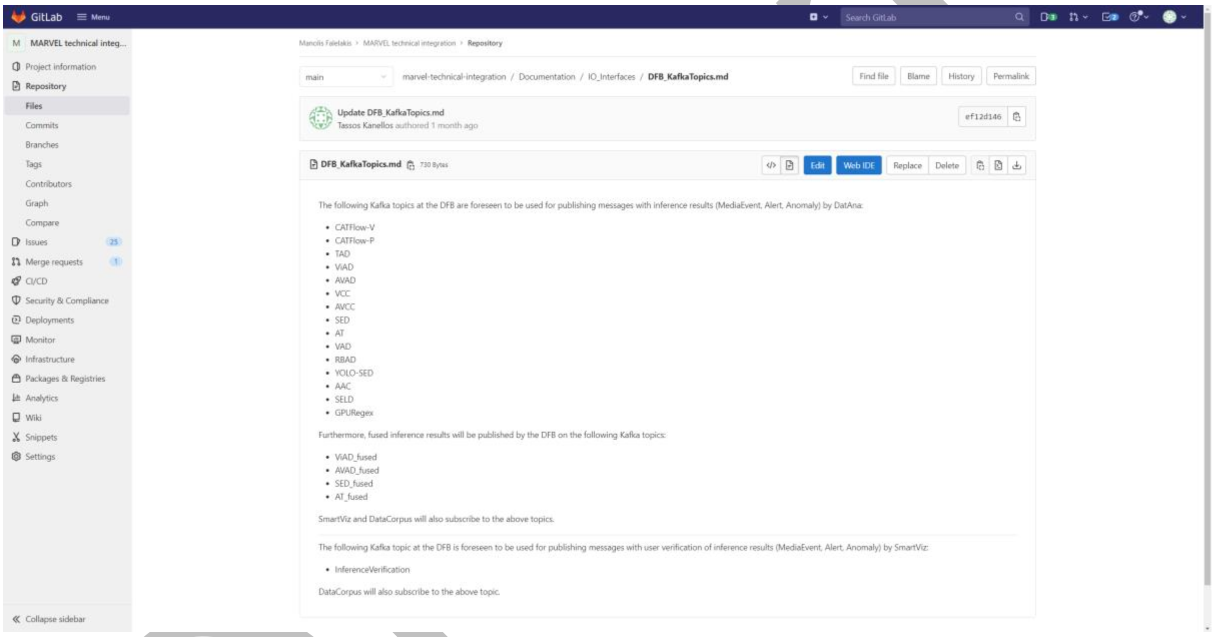
3.6 Specification documentation

The integration of multiple components in the context of the MARVEL R2 relied to a great extent on the successful inter-communication between these components. This meant that all components would need to have well-defined and well documented I/O interfaces for exchanging data with other components and that the structure of the exchanged data would need to be specified in advance and made available to the relevant components.

Following the identification of the possible pairs of components that would need to communicate with each other and the consolidation of the necessary interfaces and data models into specific interface and data model types that would serve the needs of multiple pairwise interactions, there was a need to specify these interfaces and data models in detail and maintain an always up-to-date specification documentation. The MARVEL GitLab repository was found to be highly suitable for this purpose.

Through issues registered in the ITS, MARVEL partners were requested to develop and maintain specification documents for the I/O interfaces (Figure 10) and data models (Figure 11) related to their components. The specification documents were hosted in the MARVEL GitLab repository in the form of mark-down (.md) documents. The version control features of GitLab allowed the secure updating of the documented information to align it with changing demands, while preserving a complete version history and allowing to roll back to previous versions if necessary. Using a central shared location for the maintenance of the most up-to-date specification documentation was efficient for all relevant parties to track and retrieve the necessary information. In addition, this approach allowed the use of inherent cross-references between issues listed in the ITS and specification documents.

The overall approach was established in R1 and was re-iterated for R2.



The following Kafka topics at the DFB are foreseen to be used for publishing messages with inference results (MediaEvent, Alert, Anomaly) by DatAna:

- CATFlow-V
- CATFlow-P
- TAD
- ViAD
- AVAD
- VCC
- AVCC
- SED
- AT
- VAD
- RBAD
- YOLO-SED
- AAC
- SELD
- GPURegex

Furthermore, fused inference results will be published by the DFB on the following Kafka topics:

- ViAD_fused
- AVAD_fused
- SED_fused
- AT_fused

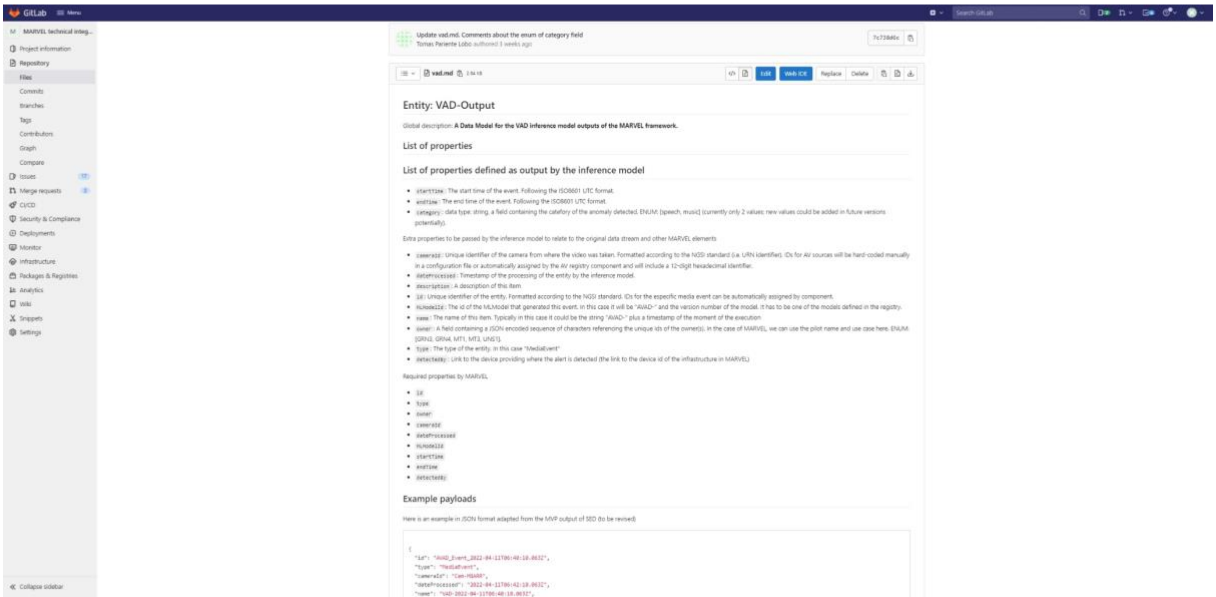
SmartViz and DataCorpus will also subscribe to the above topics.

The following Kafka topic at the DFB is foreseen to be used for publishing messages with user verification of inference results (MediaEvent, Alert, Anomaly) by SmartViz:

- InferenceVerification

DataCorpus will also subscribe to the above topic.

Figure 10: The mark-down specification document for the I/O interface between DatAna and DFB hosted in the MARVEL GitLab repository



Update vad.md. Comments about the enum of category field
Tomas Pariente Lobo authored 3 weeks ago

Entity: VAD-Output

Global description: **A Data Model for the VAD inference model outputs of the MARVEL framework.**

List of properties

List of properties defined as output by the inference model

- `startTime` : The start time of the event. Following the ISO8601 UTC format.
- `endTime` : The end time of the event. Following the ISO8601 UTC format.
- `category` : data type: string, a field containing the catefory of the anomaly detected. ENUM: [speech, music] (currently only 2 values; new values could be added in future versions potentially).

Extra properties to be passed by the inference model to relate to the original data stream and other MARVEL elements

- `cameraId` : Unique identifier of the camera from where the video was taken. Formatted according to the NGSI standard (i.e. URN identifier). IDs for AV sources will be hard-coded manually in a configuration file or automatically assigned by the AV registry component and will include a 12-digit hexadecimal identifier.
- `dateProcessed` : Timestamp of the processing of the entity by the inference model.
- `description` : A description of this item.
- `id` : Unique identifier of the entity. Formatted according to the NGSI standard. IDs for the especific media event can be automatically assigned by component.
- `MLModelId` : The id of the MLModel that generated this event. In this case it will be "AVAD-" and the version number of the model. It has to be one of the models defined in the registry.
- `name` : The name of this item. Typically in this case it could be the string "AVAD-" plus a timestamp of the moment of the execution
- `owner` : A field containing a JSON encoded sequence of characters referencing the unique ids of the owner(s). In the case of MARVEL, we can use the pilot name and use case here. ENUM [GRN3, GRN4, MT1, MT3, UNS1].
- `type` : The type of the entity. In this case "MediaEvent"
- `detectedBy` : Link to the device providing where the alert is detected (the link to the device id of the infrastructure in MARVEL)

Required properties by MARVEL

- `id`
- `type`
- `owner`
- `cameraId`
- `dateProcessed`
- `MLModelId`
- `startTime`
- `endTime`
- `detectedBy`

Example payloads

Here is an example in JSON format adapted from the MVP output of SED (to be revised)

```
{
  "id": "AVAD_Event_2022-04-11T06:40:10.063Z",
  "type": "MediaEvent",
  "cameraId": "Cam-MSARR",
  "dateProcessed": "2022-04-11T06:42:10.063Z",
  "name": "VAD-2022-04-11T06:40:10.063Z",
```

Figure 11: The mark-down specification document for the data model of the inference results produced by the VAD component, hosted in the MARVEL GitLab repository

3.7 Component deployment (CI/CD)

MARVEL adopted an iterative approach for the development and integration of the technical framework it intends to deliver. At each iteration, a functional subset of the platform was delivered for testing and demonstration purposes.

As integration advanced, the delivered platform contained an increasing number of components and services, gradually reaching the complete version of MARVEL.

The final version of the MARVEL integrated framework (R2) managed to seamlessly integrate 32 distinct components that spanned across host devices residing at the Edge, Fog, and Cloud. This was a notable progress over the achievements of R1 and the MVP, where 26 and 12 components had been integrated respectively.

These components were deployed in the form of microservices (pods) within a Kubernetes cluster. Some of these components were internally composed of multiple microservices (e.g., a single instance of the DFB is composed of 8 microservices). Additionally, in many cases, multiple instances of a single component were required to be deployed (e.g., 8 instances of DatAna MQTT and 10 instances of SED were deployed across all infrastructure nodes). At the end of R2, more than 120 services (pods) were deployed across all infrastructure nodes of the Kubernetes cluster, as part of the MARVEL integrated framework.

In addition, the components and corresponding services needed to be deployed across multiple infrastructure nodes. In R2, there were 11 nodes at the edge (6 of these part of the Kubernetes cluster), 4 nodes at the fog (3 of these part of the Kubernetes cluster) and 8 nodes (VMs) at the cloud (6 of these part of the Kubernetes cluster), distributed across the three pilots.

In order to facilitate the Continuous Integration / Continuous Delivery (CI/CD) procedure, the MARVdash component (Section 4.5.4) was used. MARVdash has been exclusively developed for the deployment of all the MARVEL components. The target deployment environment of these components comprises a Kubernetes cluster with nodes hosting containers in three layers, Edge, Fog, and Cloud.

In the context of the MVP, very few selected components had been deployed with the use of MARVdash. In that early release, most components were deployed at the host infrastructure devices using manual or semi-automated ways. In the case of R1, all components were deployed through MARVdash, which offered an automated way for the deployment procedure. Technical partners used MARVdash to provide the executables of their components in the form of docker container images as well as the necessary configuration information that controlled the deployment location and specific parameters related to the deployment of each service instance. MARVdash allowed partners to easily initialise their services in the form of pods within the available unified Kubernetes cluster that had been configured to join all available infrastructure nodes and host all services.

During R2 development and integration, MARVdash was used extensively and repetitively for deploying updated versions of component services. For example, this occurred very frequently in cases where an updated version of a component would be deployed to address issues and fix bugs that had been detected in preceding testing activities.

MARVdash serves as a dashboard that allows the creation of orchestrated containers to facilitate the deployment of E2F2C services. It offers a consolidated and user-friendly approach to managing distributed services across the entire execution-site continuum, ranging from the Edge to the Cloud. With its web-based graphical interface, MARVdash enables coordinated access to the E2F2C execution platform, facilitates the execution of services using predefined container templates, and provides seamless interaction with data collections that are automatically accessible to application containers upon launch. Moreover, MARVdash is designed to simplify user interaction with Kubernetes-based environments. It acts as a gateway for users, providing them with a landing page to access the platform. Through this user-friendly interface, users can effortlessly launch services, design workflows, request resources, and define various execution parameters.

In Kubernetes, there are two types of nodes: control-plane nodes responsible for managing the cluster, housing containers such as the API server and scheduler, and worker nodes where the containers are executed. All these nodes must be part of the same network. In MARVEL's Kubernetes cluster, the master node is installed on a virtual machine (VM) deployed on PSNC's OpenStack infrastructure. Worker nodes consist of hosts in every available layer, ranging from Edge to Fog, owned by all the MARVEL pilots. The integration of the worker nodes is facilitated by EdgeSec-VPN, which is initiated on each of them, establishing a private network as depicted in Figure 12.

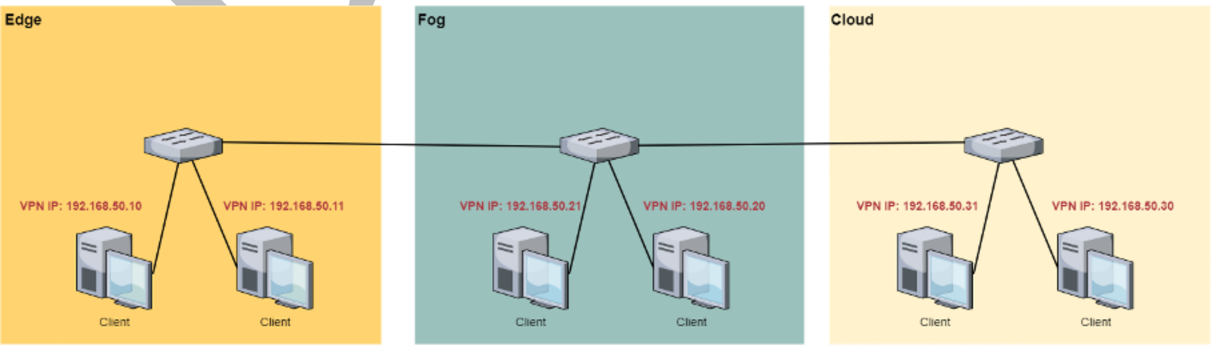


Figure 12: MARVEL Kubernetes Cluster with the use of EdgeSec VPN

Following the Kubernetes deployment paradigm, every time a new MARVEL component is to be deployed, a number of Kubernetes Pods are created with the corresponding service containers inside them. These pods are then assigned to individual worker nodes. MARVEL Kubernetes cluster utilises techniques such as 'Taints and Tolerations' and 'Node affinity' to allocate Pods to specific worker nodes, thereby achieving an E2F2C continuum. Node affinity is a characteristic of Pods that attracts them to a predefined set of nodes, either as a preference or a strict requirement. On the other hand, taints enable a node to discourage certain Pods from being scheduled onto it. To facilitate this, tolerations are applied to Pods, allowing them to be scheduled onto nodes with matching taints, but without enforcing it. By using taints and tolerations together, it is ensured that Pods are not scheduled onto unsuitable nodes. Worker nodes in the cluster are marked with one or more taints, indicating that they should not accept any Pods that cannot tolerate those taints.

MARVdash offers users a convenient method to configure and initiate services through the integration of a user-friendly service templating mechanism. This mechanism essentially utilises YAML files with customisable variables. Users can define execution parameters via the dashboard prior to deployment, and MARVdash takes care of configuring other internal platform settings, such as the location of the private container registry and external DNS name. Additionally, MARVdash efficiently handles service naming when multiple services are launched from the same template, while also providing the option for 'singleton' services that can only be deployed once per user.

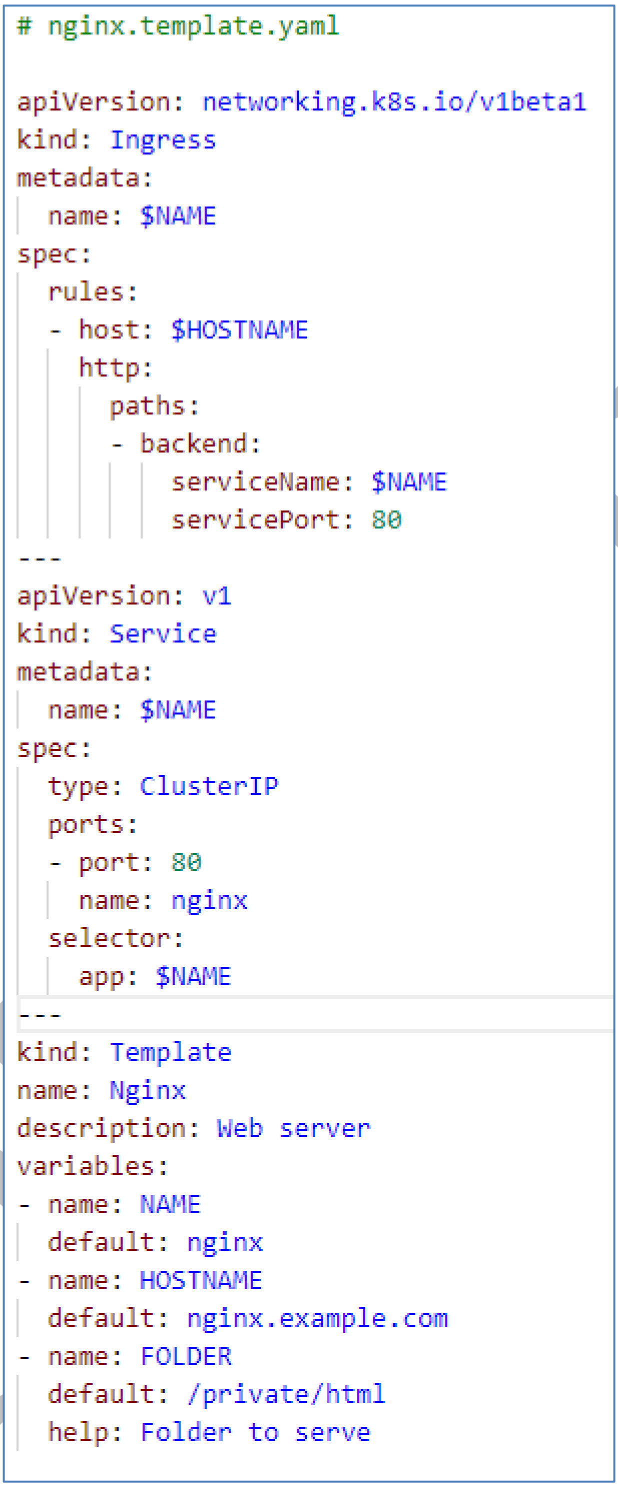
A YAML template file includes:

- one “Template” section must be included, specifying variable names, their default values, and optional help text.
- one “Service” section identified by the variable NAME. If the template comprises multiple services, MARVdash will utilise the first one by default.
- one “Ingress” section that points to the corresponding service. In such cases, the MARVdash dashboard will provide a link to the hostname assigned to the ingress, if available.

During the R1 development, MARVdash end users had to include in their components template files information about 'Taints and Tolerations' and 'Node affinity'. However, in the R2 version of MARVdash, the templates have been further simplified by removing the need for tolerations and labelling in YAML. Instead, these aspects have been replaced by a dropdown menu during the service creation process, resulting in more user-friendly and streamlined interaction.

MARVdash transforms such YAML templates to service containers taking advantage of the MARVdash private docker registry hosting container images. The newly created containers are placed into Pods and deployed to target worker nodes.

Figure 13 presents a snippet of a NGINX YAML template file. All “Template”, “Service”, and “Ingress” sections are present to allow the exposure of the NGINX functionality outside of the MARVEL Kubernetes cluster.



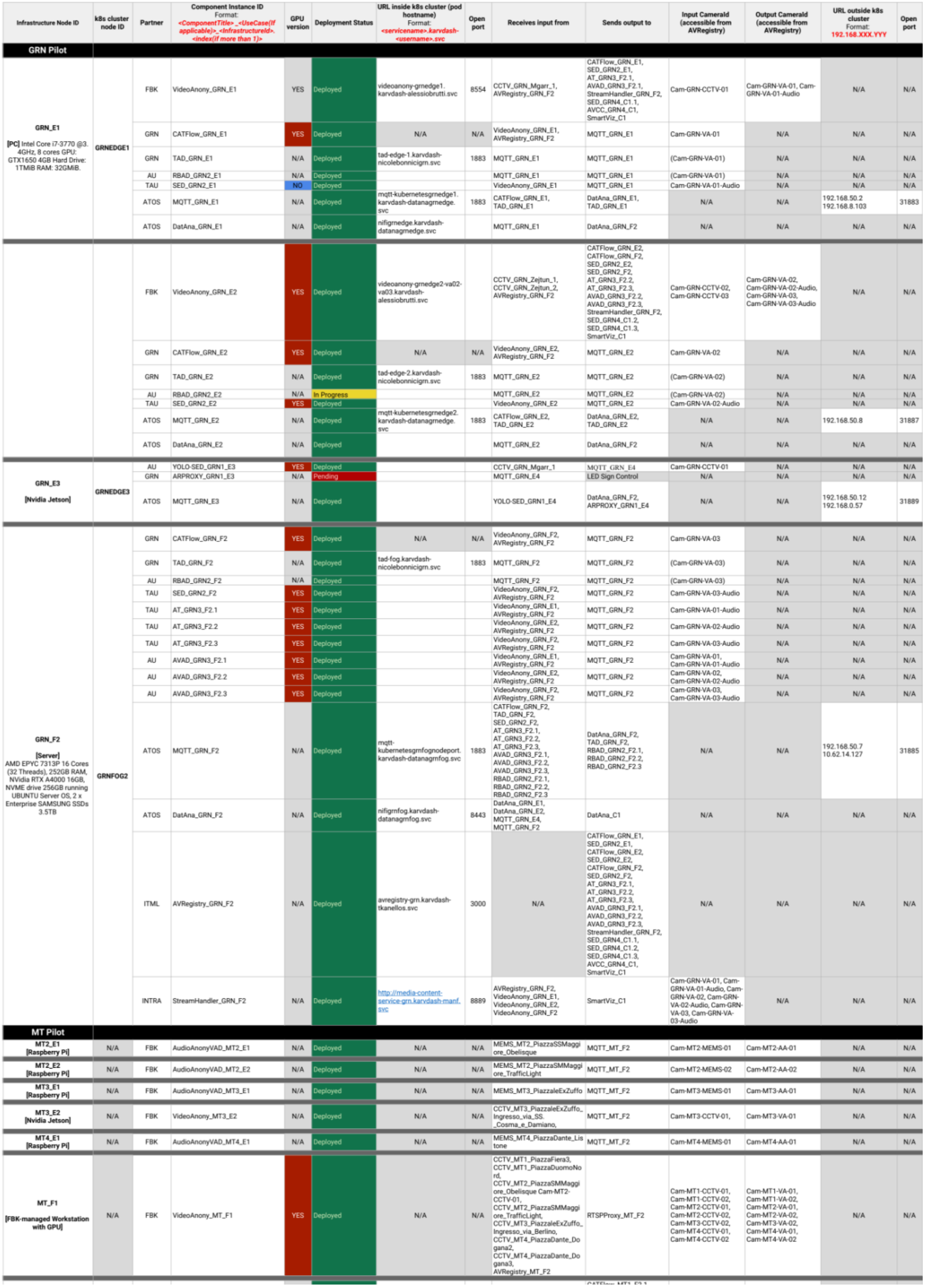
```yaml
# nginx.template.yaml

apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: $NAME
spec:
  rules:
  - host: $HOSTNAME
    http:
      paths:
      - backend:
          serviceName: $NAME
          servicePort: 80
---
apiVersion: v1
kind: Service
metadata:
  name: $NAME
spec:
  type: ClusterIP
  ports:
  - port: 80
    name: nginx
  selector:
    app: $NAME
---
kind: Template
name: Nginx
description: Web server
variables:
- name: NAME
  default: nginx
- name: HOSTNAME
  default: nginx.example.com
- name: FOLDER
  default: /private/html
  help: Folder to serve
```

Figure 13: Snippet from a NGINX YAML template file

Lastly, to facilitate the monitoring of the deployment process and document crucial information and deployment status, an online spreadsheet was created and shared with all component providers and managers involved in the pilot infrastructure (Figure 14). This spreadsheet contained detailed information on:

- Infrastructure node ids and specifications
- Component instance (service) id

- Component instance (service) deployment location (infrastructure node)
- Component instance (service) deployment status
- Component instance (service) name in the Kubernetes cluster (URL)
- Component instance (service) open port
- Component instance (service) connection to other services
- Component instance (service) input AV sources
- Component instance (service) association with Use Cases



Infrastructure Node ID	k8s cluster node ID	Partner	Component Instance ID Format: <ComponentTitle>_<UseCase(if applicable)>_<InfrastructureId>.<index(if more than 1)>	GPU version	Deployment Status	URL inside k8s cluster (pod hostname) Format: <servicename>.karvdash-<username>.svc	Open port	Receives input from	Sends output to	Input CameraId (accessible from AVRegistry)	Output CameraId (accessible from AVRegistry)	URL outside k8s cluster Format: 192.168.XXX.YYY	Open port
GRN Pilot													
GRN_E1 [PC] Intel Core i7-3770 @3.4GHz, 8 cores GPU: GTX1650 4GB Hard Drive: 1TMiB RAM: 32GMiB.	GRNEDGE1	FBK	VideoAnony_GRN_E1	YES	Deployed	videoanony-grnedge1.karvdash-alessiobrutti.svc	8554	CCTV_GRN_Mgarr_1, AVRegistry_GRN_F2	CATFlow_GRN_E1, SED_GRN2_E1, AT_GRN3_F2.1, AVAD_GRN3_F2.1, StreamHandler_GRN_F2, SED_GRN4_C1.1, AVCC_GRN4_C1, SmartViz_C1	Cam-GRN-CCTV-01	Cam-GRN-VA-01, Cam-GRN-VA-01-Audio	N/A	N/A
		GRN	CATFlow_GRN_E1	YES	Deployed	N/A	N/A	VideoAnony_GRN_E1, AVRegistry_GRN_F2	MQTT_GRN_E1	Cam-GRN-VA-01	N/A	N/A	N/A
		GRN	TAD_GRN_E1	N/A	Deployed	tad-edge-1.karvdash-nicolebonnicigrn.svc	1883	MQTT_GRN_E1	MQTT_GRN_E1	(Cam-GRN-VA-01)	N/A	N/A	N/A
		AU	RBAD_GRN2_E1	N/A	Deployed			MQTT_GRN_E1	MQTT_GRN_E1	(Cam-GRN-VA-01)	N/A	N/A	N/A
		TAU	SED_GRN2_E1	NO	Deployed			VideoAnony_GRN_E1	MQTT_GRN_E1	Cam-GRN-VA-01-Audio	N/A	N/A	N/A
		ATOS	MQTT_GRN_E1	N/A	Deployed	mqtt-kubernetesgrnedge1.karvdash-datanagrnedge.svc	1883	CATFlow_GRN_E1, TAD_GRN_E1	DatAna_GRN_E1, TAD_GRN_E1	N/A	N/A	192.168.50.2 192.168.8.103	31883
		ATOS	DatAna_GRN_E1	N/A	Deployed	nifigrnedge.karvdash-datanagrnedge.svc		MQTT_GRN_E1	DatAna_GRN_F2	N/A	N/A	N/A	N/A
		FBK	VideoAnony_GRN_E2	YES	Deployed	videoanony-grnedge2-va02-va03.karvdash-alessiobrutti.svc		CCTV_GRN_Zejtun_1, CCTV_GRN_Zejtun_2, AVRegistry_GRN_F2	CATFlow_GRN_E2, CATFlow_GRN_F2, SED_GRN2_E2, SED_GRN2_F2, AT_GRN3_F2.2, AT_GRN3_F2.3, AVAD_GRN3_F2.2, AVAD_GRN3_F2.3, StreamHandler_GRN_F2, SED_GRN4_C1.2, SED_GRN4_C1.3, SmartViz_C1	Cam-GRN-CCTV-02, Cam-GRN-CCTV-03	Cam-GRN-VA-02, Cam-GRN-VA-02-Audio, Cam-GRN-VA-03, Cam-GRN-VA-03-Audio	N/A	N/A
		GRN	CATFlow_GRN_E2	YES	Deployed	N/A	N/A	VideoAnony_GRN_E2, AVRegistry_GRN_F2	MQTT_GRN_E2	Cam-GRN-VA-02	N/A	N/A	N/A
		GRN	TAD_GRN_E2	N/A	Deployed	tad-edge-2.karvdash-nicolebonnicigrn.svc	1883	MQTT_GRN_E2	MQTT_GRN_E2	(Cam-GRN-VA-02)	N/A	N/A	N/A
		AU	RBAD_GRN2_E2	N/A	In Progress			MQTT_GRN_E2	MQTT_GRN_E2	(Cam-GRN-VA-02)	N/A	N/A	N/A
		TAU	SED_GRN2_E2	YES	Deployed			VideoAnony_GRN_E2	MQTT_GRN_E2	Cam-GRN-VA-02-Audio	N/A	N/A	N/A
		ATOS	MQTT_GRN_E2	N/A	Deployed	mqtt-kubernetesgrnedge2.karvdash-datanagrnedge.svc	1883	CATFlow_GRN_E2, TAD_GRN_E2	DatAna_GRN_E2, TAD_GRN_E2	N/A	N/A	192.168.50.8	31887
		ATOS	DatAna_GRN_E2	N/A	Deployed			MQTT_GRN_E2	DatAna_GRN_F2	N/A	N/A	N/A	N/A
GRN_E3 [Nvidia Jetson]	GRNEDGE3	AU	YOLO-SED_GRN1_E3	YES	Deployed			CCTV_GRN_Mgarr_1	MQTT_GRN_E4	Cam-GRN-CCTV-01	N/A	N/A	N/A
		GRN	ARPROXY_GRN1_E3	N/A	Pending			MQTT_GRN_E4	LED Sign Control	N/A	N/A	N/A	N/A
		ATOS	MQTT_GRN_E3	N/A	Deployed			YOLO-SED_GRN1_E4	DatAna_GRN_F2, ARPROXY_GRN1_E4	N/A	N/A	192.168.50.12 192.168.0.57	31889
GRN_F2 [Server] AMD EPYC 7313P 16 Cores (32 Threads), 252GB RAM, NVidia RTX A4000 16GB, NVME drive 256GB running UBUNTU Server OS, 2 x Enterprise SAMSUNG SSDs 3.5TB	GRNFOG2	GRN	CATFlow_GRN_F2	YES	Deployed	N/A	N/A	VideoAnony_GRN_F2, AVRegistry_GRN_F2	MQTT_GRN_F2	Cam-GRN-VA-03	N/A	N/A	N/A
		GRN	TAD_GRN_F2	N/A	Deployed	tad-fog.karvdash-nicolebonnicigrn.svc	1883	MQTT_GRN_F2	MQTT_GRN_F2	(Cam-GRN-VA-03)	N/A	N/A	N/A
		AU	RBAD_GRN2_F2	N/A	Deployed			MQTT_GRN_F2	MQTT_GRN_F2	(Cam-GRN-VA-03)	N/A	N/A	N/A
		TAU	SED_GRN2_F2	YES	Deployed			VideoAnony_GRN_F2, AVRegistry_GRN_F2	MQTT_GRN_F2	Cam-GRN-VA-03-Audio	N/A	N/A	N/A
		TAU	AT_GRN3_F2.1	YES	Deployed			VideoAnony_GRN_E1, AVRegistry_GRN_F2	MQTT_GRN_F2	Cam-GRN-VA-01-Audio	N/A	N/A	N/A
		TAU	AT_GRN3_F2.2	YES	Deployed			VideoAnony_GRN_E2, AVRegistry_GRN_F2	MQTT_GRN_F2	Cam-GRN-VA-02-Audio	N/A	N/A	N/A
		TAU	AT_GRN3_F2.3	YES	Deployed			VideoAnony_GRN_F2, AVRegistry_GRN_F2	MQTT_GRN_F2	Cam-GRN-VA-03-Audio	N/A	N/A	N/A
		AU	AVAD_GRN3_F2.1	YES	Deployed			VideoAnony_GRN_E1, AVRegistry_GRN_F2	MQTT_GRN_F2	Cam-GRN-VA-01, Cam-GRN-VA-01-Audio	N/A	N/A	N/A
		AU	AVAD_GRN3_F2.2	YES	Deployed			VideoAnony_GRN_E2, AVRegistry_GRN_F2	MQTT_GRN_F2	Cam-GRN-VA-02, Cam-GRN-VA-02-Audio	N/A	N/A	N/A
		AU	AVAD_GRN3_F2.3	YES	Deployed			VideoAnony_GRN_F2, AVRegistry_GRN_F2	MQTT_GRN_F2	Cam-GRN-VA-03, Cam-GRN-VA-03-Audio	N/A	N/A	N/A
		ATOS	MQTT_GRN_F2	N/A	Deployed	mqtt-kubernetesgrnfognodeport.karvdash-datanagrnfog.svc	1883	CATFlow_GRN_F2, TAD_GRN_F2, SED_GRN2_F2, AT_GRN3_F2.1, AT_GRN3_F2.2, AT_GRN3_F2.3, AVAD_GRN3_F2.1, AVAD_GRN3_F2.2, AVAD_GRN3_F2.3, RBAD_GRN2_F2.1, RBAD_GRN2_F2.2, RBAD_GRN2_F2.3	DatAna_GRN_F2, TAD_GRN_F2, RBAD_GRN2_F2.1, RBAD_GRN2_F2.2, RBAD_GRN2_F2.3	N/A	N/A	192.168.50.7 10.62.14.127	31885
		ATOS	DatAna_GRN_F2	N/A	Deployed	nifigrnfog.karvdash-datanagrnfog.svc	8443	DatAna_GRN_E1, DatAna_GRN_E2, MQTT_GRN_E4, MQTT_GRN_F2	DatAna_C1	N/A	N/A	N/A	N/A
		ITML	AVRegistry_GRN_F2	N/A	Deployed	avregistry-grn.karvdash-tkanellos.svc	3000	N/A	CATFlow_GRN_E1, SED_GRN2_E1, CATFlow_GRN_E2, SED_GRN2_E2, CATFlow_GRN_F2, SED_GRN2_F2, AT_GRN3_F2.1, AT_GRN3_F2.2, AT_GRN3_F2.3, AVAD_GRN3_F2.1, AVAD_GRN3_F2.2, AVAD_GRN3_F2.3, StreamHandler_GRN_F2, SED_GRN4_C1.1, SED_GRN4_C1.2, SED_GRN4_C1.3, AVCC_GRN4_C1, SmartViz_C1	N/A	N/A	N/A	N/A
		INTRA	StreamHandler_GRN_F2	N/A	Deployed	http://media-content-service-grn.karvdash-manf.svc	8889	AVRegistry_GRN_F2, VideoAnony_GRN_E1, VideoAnony_GRN_E2, VideoAnony_GRN_F2	SmartViz_C1	Cam-GRN-VA-01, Cam-GRN-VA-01-Audio, Cam-GRN-VA-02, Cam-GRN-VA-02-Audio, Cam-GRN-VA-03, Cam-GRN-VA-03-Audio	N/A	N/A	N/A
MT Pilot													
MT2_E1 [Raspberry Pi]	N/A	FBK	AudioAnonyVAD_MT2_E1	N/A	Deployed	N/A	N/A	MEMS_MT2_PiazzaSSMaggiore_Obelisque	MQTT_MT_F2	Cam-MT2-MEMS-01	Cam-MT2-AA-01	N/A	N/A
MT2_E2 [Raspberry Pi]	N/A	FBK	AudioAnonyVAD_MT2_E2	N/A	Deployed	N/A	N/A	MEMS_MT2_PiazzaSMMaggiore_TrafficLight	MQTT_MT_F2	Cam-MT2-MEMS-02	Cam-MT2-AA-02	N/A	N/A
MT3_E1 [Raspberry Pi]	N/A	FBK	AudioAnonyVAD_MT3_E1	N/A	Deployed	N/A	N/A	MEMS_MT3_PiazzaleExZuffo	MQTT_MT_F2	Cam-MT3-MEMS-01	Cam-MT3-AA-01	N/A	N/A
MT3_E2 [Nvidia Jetson]	N/A	FBK	VideoAnony_MT3_E2	N/A	Deployed	N/A	N/A	CCTV_MT3_PiazzaleExZuffo_Ingresso_via_SS._Cosma_e_Damiano,	MQTT_MT_F2	Cam-MT3-CCTV-01,	Cam-MT3-VA-01	N/A	N/A
MT4_E1 [Raspberry Pi]	N/A	FBK	AudioAnonyVAD_MT4_E1	N/A	Deployed	N/A	N/A	MEMS_MT4_PiazzaDante_Listone	MQTT_MT_F2	Cam-MT4-MEMS-01	Cam-MT4-AA-01	N/A	N/A
MT_F1 [FBK-managed Workstation with GPU]	N/A	FBK	VideoAnony_MT_F1	YES	Deployed	N/A	N/A	CCTV_MT1_PiazzaFiera3, CCTV_MT1_PiazzaDuomoNord, CCTV_MT2_PiazzaSMMaggiore_Obelisque Cam-MT2-CCTV-01, CCTV_MT2_PiazzaSMMaggiore_TrafficLight, CCTV_MT3_PiazzaleExZuffo_Ingresso_via_Berlino, CCTV_MT4_PiazzaDante_Dogana2, CCTV_MT4_PiazzaDante_Dogana3, AVRegistry_MT_F2	RTSPProxy_MT_F2	Cam-MT1-CCTV-01, Cam-MT1-CCTV-02, Cam-MT2-CCTV-01, Cam-MT2-CCTV-02, Cam-MT3-CCTV-02, Cam-MT4-CCTV-01, Cam-MT4-CCTV-02	Cam-MT1-VA-01, Cam-MT1-VA-02, Cam-MT2-VA-01, Cam-MT2-VA-02, Cam-MT3-VA-02, Cam-MT4-VA-01, Cam-MT4-VA-02	N/A	N/A

Figure 14: Part of spreadsheet for supporting the deployment process of MARVEL components in R2

3.8 Quality Assurance

In general, it is important to guarantee that each delivered increment meets high standards of quality both in terms of design and code implementation, as well as in terms of execution reliability, performance, and interoperability with other components. It was therefore necessary to devise a Quality Assurance Plan for the R2 prototype that would ensure that individual components and the complete MARVEL integrated framework meet adequate quality standards.

To that end, and in line with the procedures that had been established in R1, a high-level Quality Assurance plan was prepared for R2 that comprised the following stages:

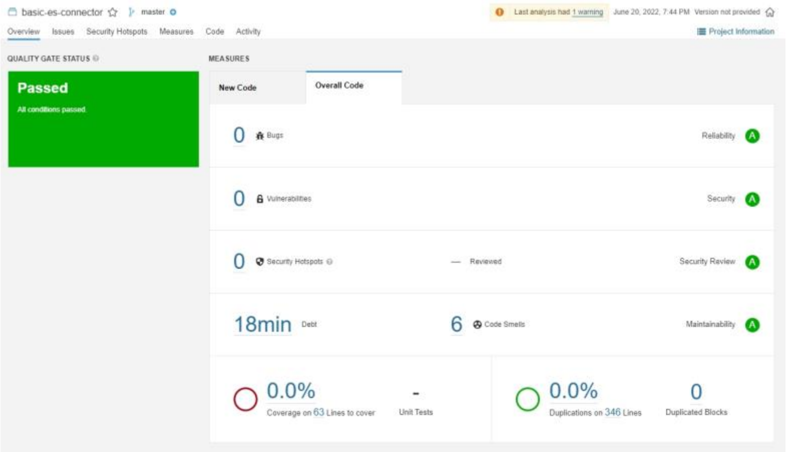
1. **Unit Testing**.
2. **Partial Integration Testing**.
3. **End-To-End Integration Testing**.
4. **Technical validation Testing**.

Quality Assurance activities took place throughout the period of R2 development and integration. In the initial stages, follow-up E2E integration testing sessions were organised to fix bugs and improve stability in the use cases that had been addressed by R1. Furthermore, regarding the new use cases that were introduced in R2, as previously described in Section 3.8, a Quality Assurance Plan was prepared and followed during the R2 integration activities. This section refers to the tests that were performed in each stage of the Quality Assurance Plan and associated results.

3.8.1 Unit Testing

Regarding unit testing, providers of MARVEL components were responsible to carry out individual analysis and testing activities of the code that was being developed to ensure that the delivered component operates as expected, i.e., test that the component can receive the foreseen input, process it and produce the expected output. A SonarQube[8] service was deployed at a VM of the PSNC HPC and integrated with the project GitLab repository (Section 3.4). Instructions were provided to partners for running the SonarQube automated static code analysis tests on their code. Partners were responsible for conducting unit testing using their own resources or the provided SonarQube service on GitLab. For example, Figure 15 illustrates an indicative SonarQube test result for a DFB service.

[8] https://www.sonarqube.org/



Last analysis had 1 warning June 20, 2022, 7:44 PM Version not provided

QUALITY GATE STATUS

Passed

All conditions passed

MEASURES

0 Bugs — Reliability A

0 Vulnerabilities — Security A

0 Security Hotspots — Reviewed — Security Review A

18min Debt — 6 Code Smells — Maintainability A

0.0% Coverage on 63 Lines to cover — - Unit Tests

0.0% Duplications on 346 Lines — 0 Duplicated Blocks

Figure 15: SonarQube automated static code analysis results for the DFB ES-Connector service

3.8.2 Partial Integration Testing

This stage was responsible for performing preliminary integration tests.

During R1 development, the Partial Integration Testing stage started in M17 and partners were initially requested to carry out tests between individual components in pairs. In order to facilitate the process, all pairs of components that were required to interact with each other were documented in a tabular format, following the structure that had been used for documenting all component pairwise I/O interfaces (presented in Figure 49, Section 5.2). In addition, each I/O Interface type (Section 5.2) was associated with a partner who would lead the relevant specification and testing activities. Each partner leading an I/O Interface type took initiatives to reach out to other partners whose components implemented the specific interface and organise partial integration tests. An online shared spreadsheet that contained an exhaustive list of the possible pairwise interactions between components and the associated integration tests was used for organisational and documentation purposes and for tracking the testing progress.

Initially, tests were conducted between services that were not necessarily deployed at the foreseen infrastructure node according to the R1 use case specifications. Instead, staging infrastructure environments that were accessible by MARVEL partners were initially employed to perform quick, provisional tests. Subsequently, as components were maturing after addressing issues that emerged from the initial tests, components gradually started to be deployed at the foreseen MARVEL infrastructure nodes via MARVdash. In parallel, partial integration tests started to involve larger groups of components that were linked together in data exchange chains as parts of the overall R1 ‘AI inference Pipeline’.

Gradually, as the number of components that were being deployed through MARVdash was increasing and the joint operation of larger component groups was being tested, it was possible to initiate the next Quality Assurance stage.

In the context of the R2 development, partial integration testing was limited since most of the I/O interfaces and APIs had already been established and mostly some revisions were only introduced. Instead, partial integration tests in R2 were mainly focused on new components that were introduced or integrated in R2 (AAC, GPURegex, EdgeSec TEE, YOLO-SED, RBAD, SELD, Arduino Proxy), for which the R1 procedure was adopted.

3.8.3 End-to-End Integration Testing

This stage is responsible for performing complete tests of the entire pipeline towards the delivery of the Release under development.

During the R1 development and integration, activities related to complete end-to-end (E2E) integration tests started in parallel to partial integration tests, but gradually the focus shifted to the E2E. In the last phase of R1 development, two (2) recurring meetings were scheduled every week in 2-hour slots. Furthermore, specialised integration groups were formed in the form of tracks for resolving issues related to specific topics, i.e., (1) AV data stream management, (2) Raw inference result management, (3) Post-processed inference result management, (4) AI Training and AI Model Repository, (5) AI Training and AI Model Repository, (6) Support on deployment with MARVdash. The partners involved in each track were focused on resolving the problems related to the track by coordinating during E2E integration tests, regularly communicating and carrying out partial integration tests.

E2E integration testing continued from the early stages of R2 integration activities, with a focus on fixing bugs and improving the stability and performance of the integrated framework in the use cases that had been addressed in R1. To that end, **twelve (12) E2E integration testing sessions were organised in the first half of R2 activities**.

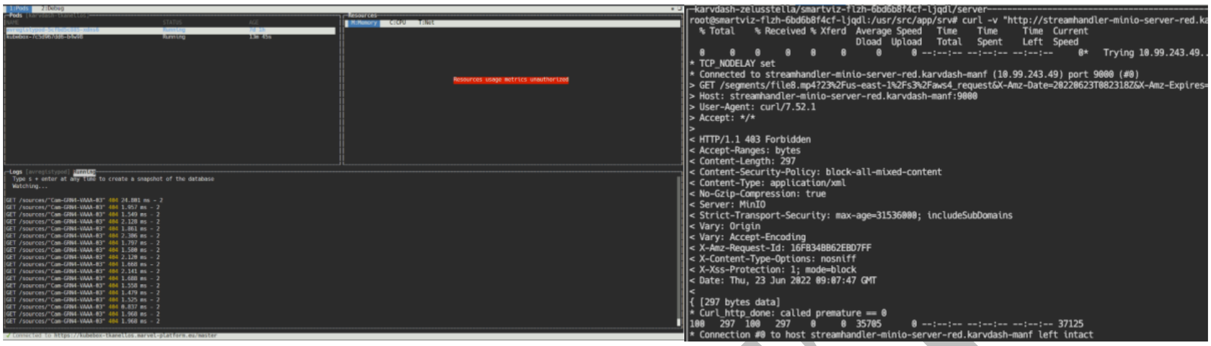
In the second half of the R2 integration period, the focus of E2E integration tests shifted from improving the use cases from R1 to testing the new use cases that were introduced in R2. The E2E integration testing sessions were realised in the same way, as described above. **Eighteen (18) E2E integration testing sessions were realised in the second half of R2**.

Each E2E integration testing session was usually addressing one or more use cases and was typically associated with a test plan that was drafted by the Integration Manager in advance. In each E2E integration test, the participation of partners that were responsible for the infrastructure or involved software components was required. Each session would typically start with a status check of the required infrastructure, AV sources and the components/services that would be required. After ensuring that all infrastructure, AV sources and services were found to be operational and properly configured, the inference pipeline steps would be tested according to the test plan. In most cases, this consisted of verifying that the necessary data was being properly generated and propagated through the inference pipeline via each intermediate node, eventually reaching the front-end application (SmartViz). In cases where problems were identified, actions were taken during the testing session to resolve them, whenever this was possible. When this occurred, it was attempted to re-arrange the order of tests and parallelise them in order to use the time of the session more efficiently (e.g., one partner focusing on resolving an issue while another continuing with other tests). During the testing session, the Integration Manager documented the test results. Identified issues that could not be resolved during the testing session would typically lead to the generation of a corresponding ticket in the MARVEL Issue Tracking System (Section 3.5).

In R2, many of the E2E testing sessions comprised stress tests of the host infrastructure at the edge, fog and cloud. As insufficient infrastructure resources (CPU, RAM, GPU, VRAM) often led to instability issues, it was of paramount importance to properly assess the capacity of infrastructure nodes to host foreseen services. During such stress tests, the load was generated by running the services that would need to be hosted on an infrastructure node in typical scenarios and the resources of the infrastructure node would be monitored. These stress tests were very useful as they drove design choices in the specification of the architecture for the R2 architecture (e.g., mapping components to available infrastructure nodes) and contributed to the optimisation of certain components in terms of resource efficiency.

During the end-to-end integration testing sessions, a series of tools were used to monitor (i) the inference pipeline and the successful completion of each intermediate step and (ii) the consumed and available infrastructure resources (e.g., CPU, RAM, GPU, VRAM, DISK). The main tools used were the following:

- **Kubebox** (command line terminal for Kubernetes). A Kubebox service was available to all MARVdash users, which could be used to run a command line terminal based on the Ubuntu OS and monitor the status and activity of deployed services (Figure 16).

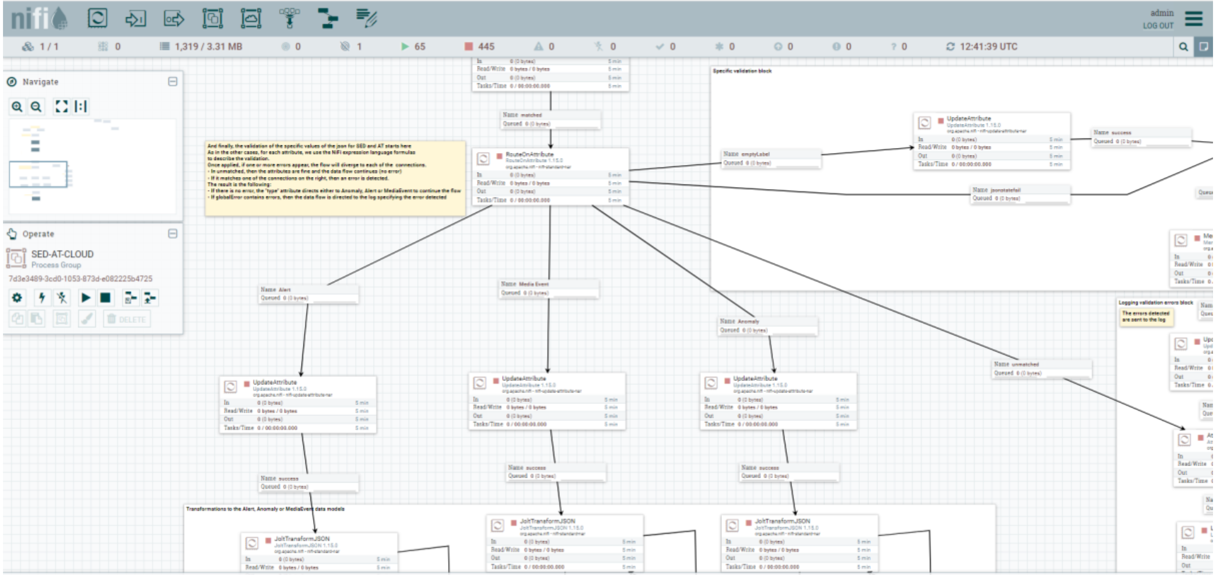


```
Pods [karvdash-tkanellos]
NAME                               STATUS     AGE
avregistrypod-5c7bd5c885-xdns6     Running    7d 1h
kubebox-7c5d967dd6-b4w98           Running    13m 45s

Logs [avregistrypod] Running
 Type s + enter at any time to create a snapshot of the database
 Watching...

GET /sources/"Cam-GRN4-VAAA-03" 404 24.881 ms - 2
GET /sources/"Cam-GRN4-VAAA-03" 404 1.957 ms - 2
GET /sources/"Cam-GRN4-VAAA-03" 404 1.549 ms - 2
GET /sources/"Cam-GRN4-VAAA-03" 404 2.128 ms - 2
GET /sources/"Cam-GRN4-VAAA-03" 404 1.861 ms - 2
GET /sources/"Cam-GRN4-VAAA-03" 404 2.386 ms - 2
GET /sources/"Cam-GRN4-VAAA-03" 404 1.797 ms - 2
GET /sources/"Cam-GRN4-VAAA-03" 404 1.588 ms - 2
GET /sources/"Cam-GRN4-VAAA-03" 404 2.128 ms - 2
GET /sources/"Cam-GRN4-VAAA-03" 404 1.668 ms - 2
GET /sources/"Cam-GRN4-VAAA-03" 404 2.141 ms - 2
GET /sources/"Cam-GRN4-VAAA-03" 404 1.688 ms - 2
GET /sources/"Cam-GRN4-VAAA-03" 404 1.558 ms - 2
GET /sources/"Cam-GRN4-VAAA-03" 404 1.479 ms - 2
GET /sources/"Cam-GRN4-VAAA-03" 404 1.525 ms - 2
GET /sources/"Cam-GRN4-VAAA-03" 404 0.837 ms - 2
GET /sources/"Cam-GRN4-VAAA-03" 404 1.968 ms - 2
GET /sources/"Cam-GRN4-VAAA-03" 404 1.968 ms - 2
✓ Connected to https://kubebox-tkanellos.marvel-platform.eu/master
```

```
-karvdash-zelusstella/smartviz-flzh-6bd6b8f4cf-ljqdl/server-----------------------
root@smartviz-flzh-6bd6b8f4cf-ljqdl:/usr/src/app/srv# curl -v "http://streamhandler-minio-server-red.ka
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0*   Trying 10.99.243.49..
* TCP_NODELAY set
* Connected to streamhandler-minio-server-red.karvdash-manf (10.99.243.49) port 9000 (#0)
> GET /segments/file8.mp4?23%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-Date=20220623T082318Z&X-Amz-Expires=
> Host: streamhandler-minio-server-red.karvdash-manf:9000
> User-Agent: curl/7.52.1
> Accept: */*
>
< HTTP/1.1 403 Forbidden
< Accept-Ranges: bytes
< Content-Length: 297
< Content-Security-Policy: block-all-mixed-content
< Content-Type: application/xml
< No-Gzip-Compression: true
< Server: MinIO
< Strict-Transport-Security: max-age=31536000; includeSubDomains
< Vary: Origin
< Vary: Accept-Encoding
< X-Amz-Request-Id: 16FB34BB62EBD7FF
< X-Content-Type-Options: nosniff
< X-Xss-Protection: 1; mode=block
< Date: Thu, 23 Jun 2022 09:07:47 GMT
<
{ [297 bytes data]
* Curl_http_done: called premature == 0
100   297  100   297    0     0  35705      0 --:--:-- --:--:-- --:--:-- 37125
* Connection #0 to host streamhandler-minio-server-red.karvdash-manf left intact
```

Figure 16: Kubebox terminal and associated Ubuntu OS command line

- **DatAna NiFi GUI**. Being an integral part of DatAna, this tool was used to monitor and configure the data flows of raw AI inference results that were published to the DatAna MQTT brokers, perform data validation tasks, transformation into the SDM-compliant AI Inference Result data models and relay data to higher DatAna layers and ultimately to the DFB (Figure 17).



And finally, the validation of the specific values of the json for SED and AT starts here
As in the other cases, for each attribute, we use the NiFi expression language formulas
to describe the validation.
Once applied, if one or more errors appear, the flow will diverge to each of the connections.
- In unmatched, then the attributes are fine and the data flow continues (no error)
- If it matches one of the connections on the right, then an error is detected.
The result is the following:
- If there is no error, the 'type' attribute directs either to Anomaly, Alert or MediaEvent to continue the flow
- If globalError contains errors, then the data flow is directed to the log specifying the error detected

Figure 17: DatAna NiFi GUI

- **DFB Kafka GUI for monitoring** (Grafana). The DFB Kafka monitoring tool built with Grafana was used to monitor the traffic on Kafka topics and other performance parameters of the DFB Kafka brokers (Figure 18).

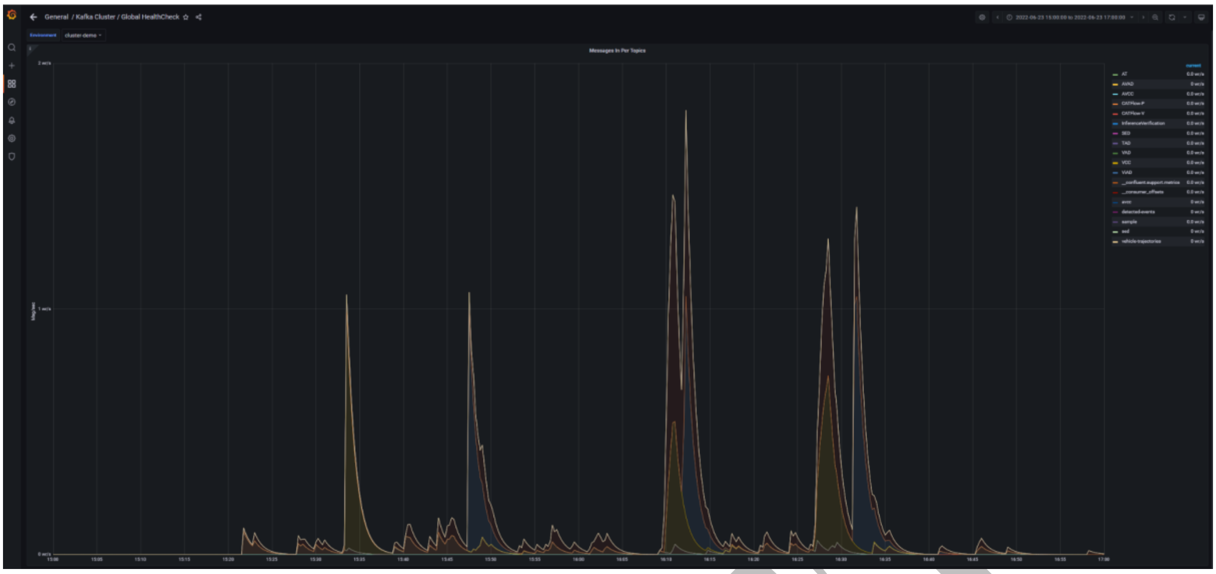
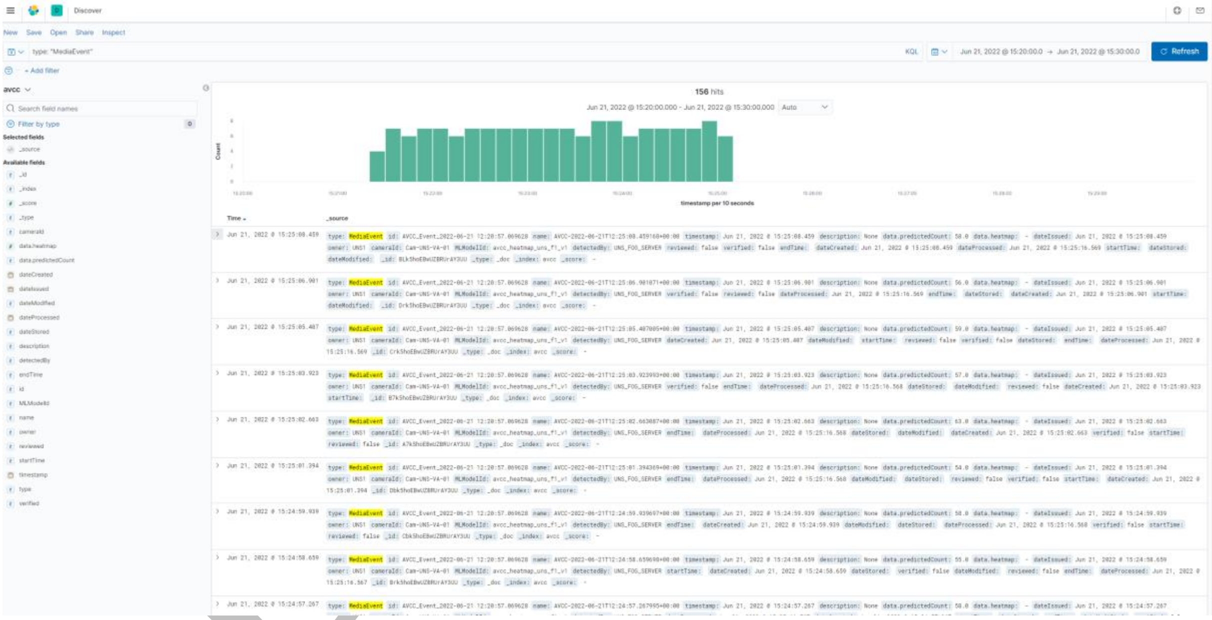


Figure 18: DFB Kafka topic traffic monitoring

- **DFB ES GUI for monitoring** (Kibana). A Kibana service was used to monitor the data that was being ingested into the DFB ES repository (Figure 19).

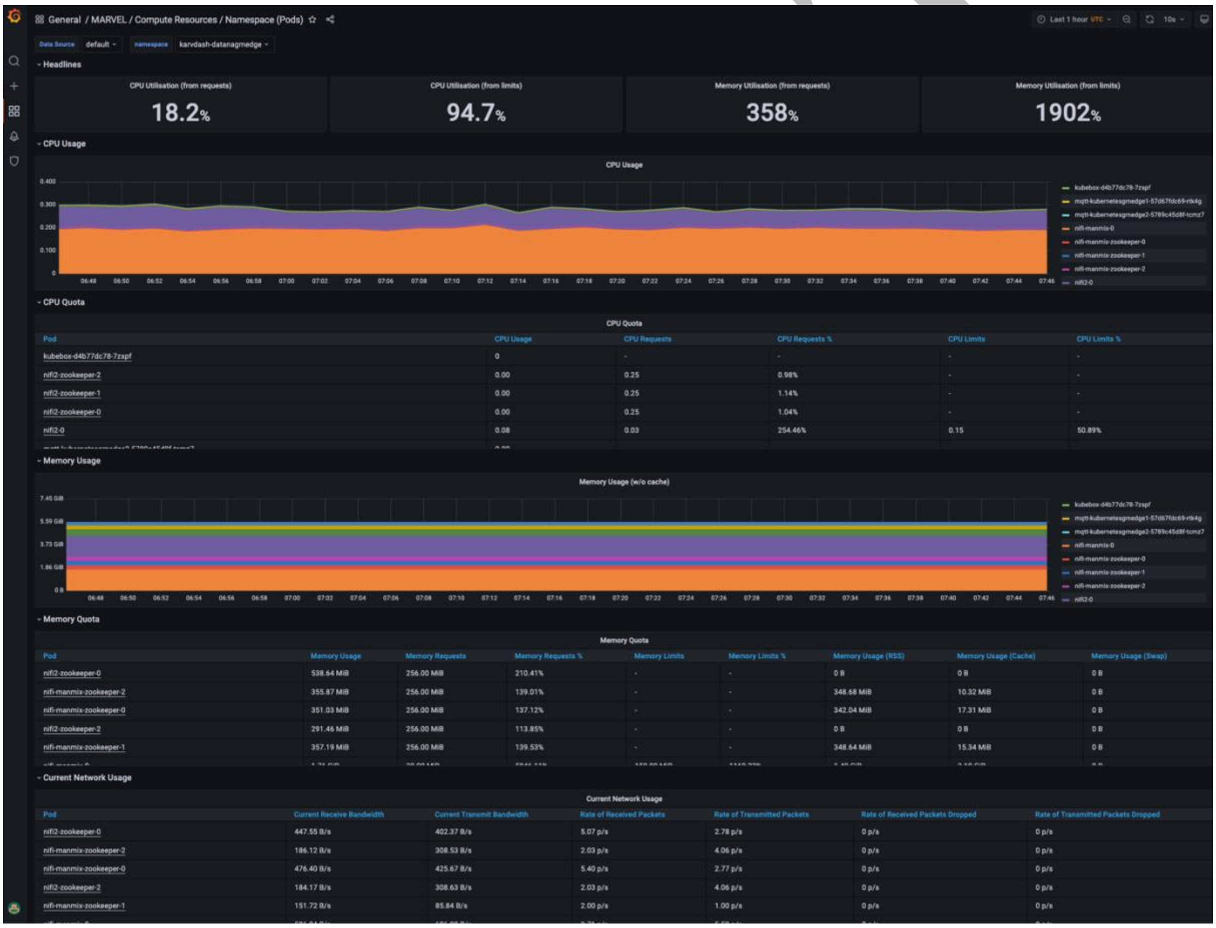


```
Jun 21, 2022 @ 15:25:08.459  type: MediaEvent id: AVCC_Event_2022-06-21 12:20:57.069628 name: AVCC-2022-06-21T12:25:08.459168+00:00 timestamp: Jun 21, 2022 @ 15:25:08.459 description: None data.predictedCount: 58.0 data.heatmap: - dateIssued: Jun 21, 2022 @ 15:25:08.459
                             owner: UNS1 cameraId: Cam-UNS-VA-01 MLModelId: avcc_heatmap_uns_f1_v1 detectedBy: UNS_FOG_SERVER reviewed: false verified: false endTime:  dateCreated: Jun 21, 2022 @ 15:25:08.459 dateProcessed: Jun 21, 2022 @ 15:25:16.569 startTime:  dateStored:
                             dateModified:  _id: BLkShoEBwUZBRUrAY3UU _type: _doc _index: avcc _score: -
Jun 21, 2022 @ 15:25:06.901  type: MediaEvent id: AVCC_Event_2022-06-21 12:20:57.069628 name: AVCC-2022-06-21T12:25:06.901871+00:00 timestamp: Jun 21, 2022 @ 15:25:06.901 description: None data.predictedCount: 56.0 data.heatmap: - dateIssued: Jun 21, 2022 @ 15:25:06.901
                             owner: UNS1 cameraId: Cam-UNS-VA-01 MLModelId: avcc_heatmap_uns_f1_v1 detectedBy: UNS_FOG_SERVER verified: false reviewed: false dateProcessed: Jun 21, 2022 @ 15:25:16.569 endTime:  dateStored:  dateCreated: Jun 21, 2022 @ 15:25:06.901 startTime:
                             dateModified:  _id: DrkShoEBwUZBRUrAY3UU _type: _doc _index: avcc _score: -
Jun 21, 2022 @ 15:25:05.487  type: MediaEvent id: AVCC_Event_2022-06-21 12:20:57.069628 name: AVCC-2022-06-21T12:25:05.487085+00:00 timestamp: Jun 21, 2022 @ 15:25:05.487 description: None data.predictedCount: 59.0 data.heatmap: - dateIssued: Jun 21, 2022 @ 15:25:05.487
                             owner: UNS1 cameraId: Cam-UNS-VA-01 MLModelId: avcc_heatmap_uns_f1_v1 detectedBy: UNS_FOG_SERVER dateCreated: Jun 21, 2022 @ 15:25:05.487 dateModified:  startTime:  reviewed: false verified: false dateStored:  endTime:  dateProcessed: Jun 21, 2022 @
                             15:25:16.569 _id: CrkShoEBwUZBRUrAY3UU _type: _doc _index: avcc _score: -
Jun 21, 2022 @ 15:25:03.923  type: MediaEvent id: AVCC_Event_2022-06-21 12:20:57.069628 name: AVCC-2022-06-21T12:25:03.923993+00:00 timestamp: Jun 21, 2022 @ 15:25:03.923 description: None data.predictedCount: 57.0 data.heatmap: - dateIssued: Jun 21, 2022 @ 15:25:03.923
                             owner: UNS1 cameraId: Cam-UNS-VA-01 MLModelId: avcc_heatmap_uns_f1_v1 detectedBy: UNS_FOG_SERVER verified: false endTime:  dateProcessed: Jun 21, 2022 @ 15:25:16.568 dateStored:  dateModified:  reviewed: false dateCreated: Jun 21, 2022 @ 15:25:03.923
                             startTime:  _id: B7kShoEBwUZBRUrAY3UU _type: _doc _index: avcc _score: -
Jun 21, 2022 @ 15:25:02.663  type: MediaEvent id: AVCC_Event_2022-06-21 12:20:57.069628 name: AVCC-2022-06-21T12:25:02.663087+00:00 timestamp: Jun 21, 2022 @ 15:25:02.663 description: None data.predictedCount: 63.0 data.heatmap: - dateIssued: Jun 21, 2022 @ 15:25:02.663
                             owner: UNS1 cameraId: Cam-UNS-VA-01 MLModelId: avcc_heatmap_uns_f1_v1 detectedBy: UNS_FOG_SERVER endTime:  dateProcessed: Jun 21, 2022 @ 15:25:16.568 dateStored:  dateModified:  dateCreated: Jun 21, 2022 @ 15:25:02.663 verified: false startTime:
                             reviewed: false _id: A7kShoEBwUZBRUrAY3UU _type: _doc _index: avcc _score: -
Jun 21, 2022 @ 15:25:01.394  type: MediaEvent id: AVCC_Event_2022-06-21 12:20:57.069628 name: AVCC-2022-06-21T12:25:01.394369+00:00 timestamp: Jun 21, 2022 @ 15:25:01.394 description: None data.predictedCount: 54.0 data.heatmap: - dateIssued: Jun 21, 2022 @ 15:25:01.394
                             owner: UNS1 cameraId: Cam-UNS-VA-01 MLModelId: avcc_heatmap_uns_f1_v1 detectedBy: UNS_FOG_SERVER endTime:  dateProcessed: Jun 21, 2022 @ 15:25:16.568 dateModified:  dateStored:  reviewed: false verified: false startTime:  dateCreated: Jun 21, 2022 @
                             15:25:01.394 _id: DbkShoEBwUZBRUrAY3UU _type: _doc _index: avcc _score: -
Jun 21, 2022 @ 15:24:59.939  type: MediaEvent id: AVCC_Event_2022-06-21 12:20:57.069628 name: AVCC-2022-06-21T12:24:59.939697+00:00 timestamp: Jun 21, 2022 @ 15:24:59.939 description: None data.predictedCount: 58.0 data.heatmap: - dateIssued: Jun 21, 2022 @ 15:24:59.939
                             owner: UNS1 cameraId: Cam-UNS-VA-01 MLModelId: avcc_heatmap_uns_f1_v1 detectedBy: UNS_FOG_SERVER endTime:  dateCreated: Jun 21, 2022 @ 15:24:59.939 dateModified:  dateStored:  dateProcessed: Jun 21, 2022 @ 15:25:16.568 verified: false startTime:
                             reviewed: false _id: CbkShoEBwUZBRUrAY3UU _type: _doc _index: avcc _score: -
Jun 21, 2022 @ 15:24:58.659  type: MediaEvent id: AVCC_Event_2022-06-21 12:20:57.069628 name: AVCC-2022-06-21T12:24:58.659690+00:00 timestamp: Jun 21, 2022 @ 15:24:58.659 description: None data.predictedCount: 55.0 data.heatmap: - dateIssued: Jun 21, 2022 @ 15:24:58.659
                             owner: UNS1 cameraId: Cam-UNS-VA-01 MLModelId: avcc_heatmap_uns_f1_v1 detectedBy: UNS_FOG_SERVER startTime:  dateCreated: Jun 21, 2022 @ 15:24:58.659 dateStored:  verified: false dateModified:  reviewed: false endTime:  dateProcessed: Jun 21, 2022 @
                             15:25:16.567 _id: BrkShoEBwUZBRUrAY3UU _type: _doc _index: avcc _score: -
Jun 21, 2022 @ 15:24:57.267  type: MediaEvent id: AVCC_Event_2022-06-21 12:20:57.069628 name: AVCC-2022-06-21T12:24:57.267995+00:00 timestamp: Jun 21, 2022 @ 15:24:57.267 description: None data.predictedCount: 58.0 data.heatmap: - dateIssued: Jun 21, 2022 @ 15:24:57.267
```

Figure 19: DFB Elastic Search GUI (Kibana)

- **SmartViz**. Being the main front-end component of the MARVEL framework, SmartViz was used for checking the quality of the data consumed through the use of the dedicated GUI widgets it incorporates.
- **MARVdash monitoring tool**. During R2 activities, a dedicated monitoring tool for MARVdash was developed that acted as a central portal for monitoring many aspects of the running MARVEL framework through a GUI. This tool was based on the

Prometheus[9], Grafana[10], and Loki[11] technologies and offered multiple pre-built dashboards that visualised different aspects of the MARVEL framework. For example, some of the dashboards refer to monitoring the computational resources consumed by each running Kubernetes pod, organised by infrastructure node or by user namespace (Figure 20). Another notable example is a dashboard for monitoring the log output that is generated by any individual component instance that is running as a pod in the Kubernetes cluster via MARVdash (Figure 21). This was particularly useful for monitoring the internal status of individual components and for debugging purposes. Another very useful dashboard presented the status of multiple DatAna MQTT brokers simultaneously (Figure 22). Since MQTT brokers were deployed on most infrastructure nodes and they all performed a pivotal role in the collection and propagation of inference results in the inference pipeline, it was important to be able to quickly understand if any of these brokers or a component connected to them was facing an issue.



CPU Utilisation (from requests)	CPU Utilisation (from limits)	Memory Utilisation (from requests)	Memory Utilisation (from limits)
18.2%	94.7%	358%	1902%

CPU Quota

Pod	CPU Usage	CPU Requests	CPU Requests %	CPU Limits	CPU Limits %
kubebox-d4b77dc78-7zspf	0	-	-	-	-
nifi2-zookeeper-2	0.00	0.25	0.98%	-	-
nifi2-zookeeper-1	0.00	0.25	1.14%	-	-
nifi2-zookeeper-0	0.00	0.25	1.04%	-	-
nifi2-0	0.08	0.03	254.46%	0.15	50.89%

Memory Quota

Pod	Memory Usage	Memory Requests	Memory Requests %	Memory Limits	Memory Limits %	Memory Usage (RSS)	Memory Usage (Cache)	Memory Usage (Swap)
nifi2-zookeeper-0	538.64 MiB	256.00 MiB	210.41%	-	-	0 B	0 B	0 B
nifi-manmix-zookeeper-2	355.87 MiB	256.00 MiB	139.01%	-	-	348.68 MiB	10.32 MiB	0 B
nifi-manmix-zookeeper-0	351.03 MiB	256.00 MiB	137.12%	-	-	342.04 MiB	17.31 MiB	0 B
nifi2-zookeeper-2	291.46 MiB	256.00 MiB	113.85%	-	-	0 B	0 B	0 B
nifi-manmix-zookeeper-1	357.19 MiB	256.00 MiB	139.53%	-	-	348.64 MiB	15.34 MiB	0 B

Current Network Usage

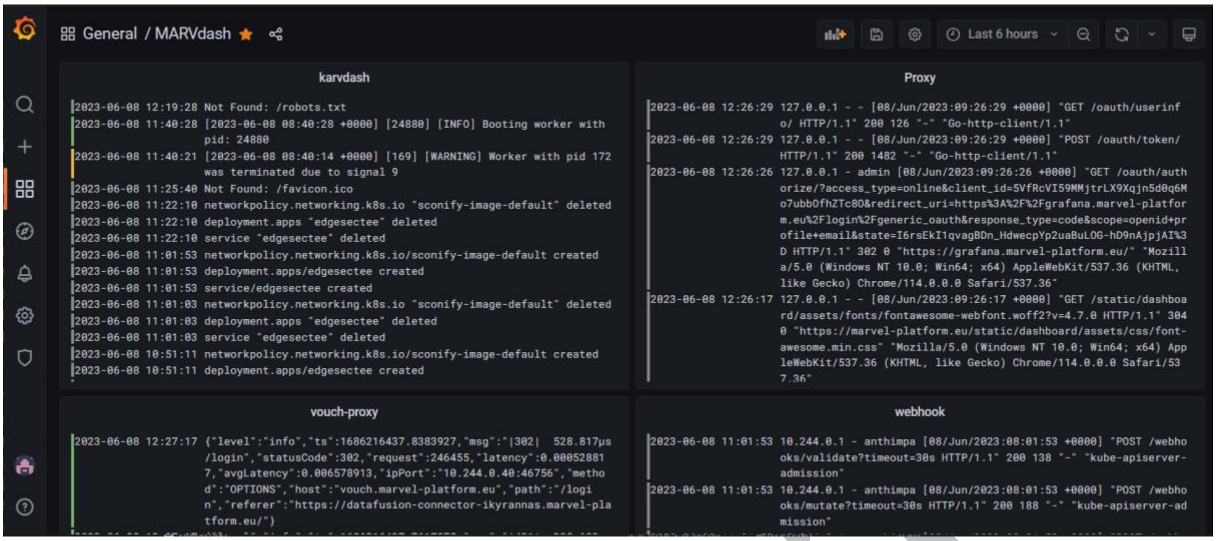
Pod	Current Receive Bandwidth	Current Transmit Bandwidth	Rate of Received Packets	Rate of Transmitted Packets	Rate of Received Packets Dropped	Rate of Transmitted Packets Dropped
nifi2-zookeeper-0	447.55 B/s	402.37 B/s	5.07 p/s	2.78 p/s	0 p/s	0 p/s
nifi-manmix-zookeeper-2	186.12 B/s	308.53 B/s	2.03 p/s	4.06 p/s	0 p/s	0 p/s
nifi-manmix-zookeeper-0	476.40 B/s	425.67 B/s	5.40 p/s	2.77 p/s	0 p/s	0 p/s
nifi2-zookeeper-2	184.17 B/s	308.63 B/s	2.03 p/s	4.06 p/s	0 p/s	0 p/s
nifi-manmix-zookeeper-1	151.72 B/s	85.84 B/s	2.00 p/s	1.00 p/s	0 p/s	0 p/s

Figure 20: MARVdash monitoring dashboard: Compute Resources per namespace

[9] https://prometheus.io/

[10] https://grafana.com/

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



karvdash

```
2023-06-08 12:19:28 Not Found: /robots.txt
2023-06-08 11:40:28 [2023-06-08 08:40:28 +0000] [24880] [INFO] Booting worker with
                    pid: 24880
2023-06-08 11:40:21 [2023-06-08 08:40:14 +0000] [169] [WARNING] Worker with pid 172
                    was terminated due to signal 9
2023-06-08 11:25:40 Not Found: /favicon.ico
2023-06-08 11:22:10 networkpolicy.networking.k8s.io "sconify-image-default" deleted
2023-06-08 11:22:10 deployment.apps "edgesectee" deleted
2023-06-08 11:22:10 service "edgesectee" deleted
2023-06-08 11:01:53 networkpolicy.networking.k8s.io/sconify-image-default created
2023-06-08 11:01:53 deployment.apps/edgesectee created
2023-06-08 11:01:53 service/edgesectee created
2023-06-08 11:01:03 networkpolicy.networking.k8s.io "sconify-image-default" deleted
2023-06-08 11:01:03 deployment.apps "edgesectee" deleted
2023-06-08 11:01:03 service "edgesectee" deleted
2023-06-08 10:51:11 networkpolicy.networking.k8s.io/sconify-image-default created
2023-06-08 10:51:11 deployment.apps/edgesectee created
```

Proxy

```
2023-06-08 12:26:29 127.0.0.1 - - [08/Jun/2023:09:26:29 +0000] "GET /oauth/userinf
                    o/ HTTP/1.1" 200 126 "-" "Go-http-client/1.1"
2023-06-08 12:26:29 127.0.0.1 - - [08/Jun/2023:09:26:29 +0000] "POST /oauth/token/
                    HTTP/1.1" 200 1482 "-" "Go-http-client/1.1"
2023-06-08 12:26:26 127.0.0.1 - admin [08/Jun/2023:09:26:26 +0000] "GET /oauth/auth
                    orize/?access_type=online&client_id=5VfRcVI59MMjtrLX9Xqjn5d0q6M
                    o7ubbOfhZTc8O&redirect_uri=https%3A%2F%2Fgrafana.marvel-platfor
                    m.eu%2Flogin%2Fgeneric_oauth&response_type=code&scope=openid+pr
                    ofile+email&state=I6rsEkI1qvagBDn_HdwecpYp2uaBuLOG-hD9nAjpjAI%3
                    D HTTP/1.1" 302 0 "https://grafana.marvel-platform.eu/" "Mozill
                    a/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,
                    like Gecko) Chrome/114.0.0.0 Safari/537.36"
2023-06-08 12:26:17 127.0.0.1 - - [08/Jun/2023:09:26:17 +0000] "GET /static/dashboa
                    rd/assets/fonts/fontawesome-webfont.woff2?v=4.7.0 HTTP/1.1" 304
                    0 "https://marvel-platform.eu/static/dashboard/assets/css/font-
                    awesome.min.css" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) App
                    leWebKit/537.36 (KHTML, like Gecko) Chrome/114.0.0.0 Safari/53
                    7.36"
```

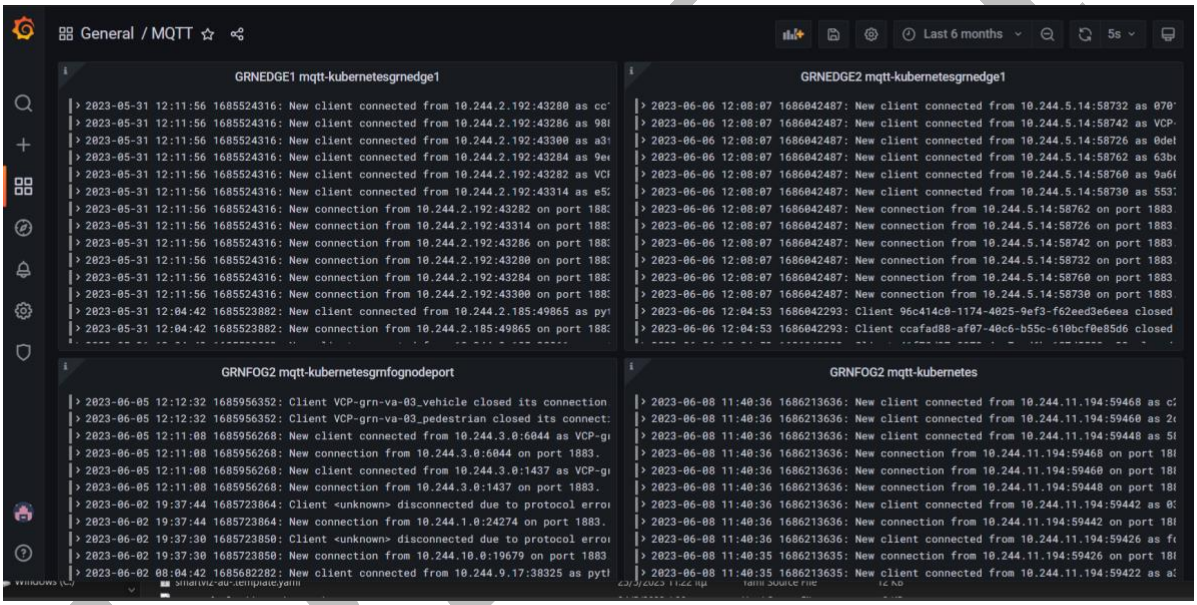
vouch-proxy

```
2023-06-08 12:27:17 {"level":"info","ts":1686216437.8383927,"msg":"|302|  528.817µs
                    /login","statusCode":302,"request":246455,"latency":0.00052881
                    7,"avgLatency":0.006578913,"ipPort":"10.244.0.40:46756","metho
                    d":"OPTIONS","host":"vouch.marvel-platform.eu","path":"/logi
                    n","referer":"https://datafusion-connector-ikyrannas.marvel-pla
                    tform.eu/"}
```

webhook

```
2023-06-08 11:01:53 10.244.0.1 - anthimpa [08/Jun/2023:08:01:53 +0000] "POST /webho
                    oks/validate?timeout=30s HTTP/1.1" 200 138 "-" "kube-apiserver-
                    admission"
2023-06-08 11:01:53 10.244.0.1 - anthimpa [08/Jun/2023:08:01:53 +0000] "POST /webho
                    oks/mutate?timeout=30s HTTP/1.1" 200 188 "-" "kube-apiserver-ad
                    mission"
```

Figure 21: MARVdash monitoring dashboard: Service Logs



GRNEDGE1 mqtt-kubernetesgrnedge1

```
> 2023-05-31 12:11:56 1685524316: New client connected from 10.244.2.192:43280 as cc
> 2023-05-31 12:11:56 1685524316: New client connected from 10.244.2.192:43286 as 98
> 2023-05-31 12:11:56 1685524316: New client connected from 10.244.2.192:43300 as a3
> 2023-05-31 12:11:56 1685524316: New client connected from 10.244.2.192:43284 as 9e
> 2023-05-31 12:11:56 1685524316: New client connected from 10.244.2.192:43282 as VC
> 2023-05-31 12:11:56 1685524316: New client connected from 10.244.2.192:43314 as e5
> 2023-05-31 12:11:56 1685524316: New connection from 10.244.2.192:43282 on port 188
> 2023-05-31 12:11:56 1685524316: New connection from 10.244.2.192:43314 on port 188
> 2023-05-31 12:11:56 1685524316: New connection from 10.244.2.192:43286 on port 188
> 2023-05-31 12:11:56 1685524316: New connection from 10.244.2.192:43280 on port 188
> 2023-05-31 12:11:56 1685524316: New connection from 10.244.2.192:43284 on port 188
> 2023-05-31 12:11:56 1685524316: New connection from 10.244.2.192:43300 on port 188
> 2023-05-31 12:04:42 1685523882: New client connected from 10.244.2.185:49865 as py
> 2023-05-31 12:04:42 1685523882: New connection from 10.244.2.185:49865 on port 188
```

GRNEDGE2 mqtt-kubernetesgrnedge1

```
> 2023-06-06 12:08:07 1686042487: New client connected from 10.244.5.14:58732 as 070
> 2023-06-06 12:08:07 1686042487: New client connected from 10.244.5.14:58742 as VCP-
> 2023-06-06 12:08:07 1686042487: New client connected from 10.244.5.14:58726 as 0de
> 2023-06-06 12:08:07 1686042487: New client connected from 10.244.5.14:58762 as 63b
> 2023-06-06 12:08:07 1686042487: New client connected from 10.244.5.14:58760 as 9a6
> 2023-06-06 12:08:07 1686042487: New client connected from 10.244.5.14:58730 as 553
> 2023-06-06 12:08:07 1686042487: New connection from 10.244.5.14:58762 on port 1883.
> 2023-06-06 12:08:07 1686042487: New connection from 10.244.5.14:58726 on port 1883.
> 2023-06-06 12:08:07 1686042487: New connection from 10.244.5.14:58742 on port 1883.
> 2023-06-06 12:08:07 1686042487: New connection from 10.244.5.14:58732 on port 1883.
> 2023-06-06 12:08:07 1686042487: New connection from 10.244.5.14:58760 on port 1883.
> 2023-06-06 12:08:07 1686042487: New connection from 10.244.5.14:58730 on port 1883.
> 2023-06-06 12:04:53 1686042293: Client 96c414c0-1174-4025-9ef3-f62eed3e6eea closed
> 2023-06-06 12:04:53 1686042293: Client ccafad88-af07-40c6-b55c-610bcf0e85d6 closed
```

GRNFOG2 mqtt-kubernetesgrnfognodeport

```
> 2023-06-05 12:12:32 1685956352: Client VCP-grn-va-03_vehicle closed its connection
> 2023-06-05 12:12:32 1685956352: Client VCP-grn-va-03_pedestrian closed its connect
> 2023-06-05 12:11:08 1685956268: New client connected from 10.244.3.0:6044 as VCP-g
> 2023-06-05 12:11:08 1685956268: New connection from 10.244.3.0:6044 on port 1883.
> 2023-06-05 12:11:08 1685956268: New client connected from 10.244.3.0:1437 as VCP-g
> 2023-06-05 12:11:08 1685956268: New connection from 10.244.3.0:1437 on port 1883.
> 2023-06-02 19:37:44 1685723864: Client <unknown> disconnected due to protocol erro
> 2023-06-02 19:37:44 1685723864: New connection from 10.244.1.0:24274 on port 1883.
> 2023-06-02 19:37:30 1685723850: Client <unknown> disconnected due to protocol erro
> 2023-06-02 19:37:30 1685723850: New connection from 10.244.10.0:19679 on port 1883
> 2023-06-02 08:04:42 1685682282: New client connected from 10.244.9.17:38325 as pyt
```

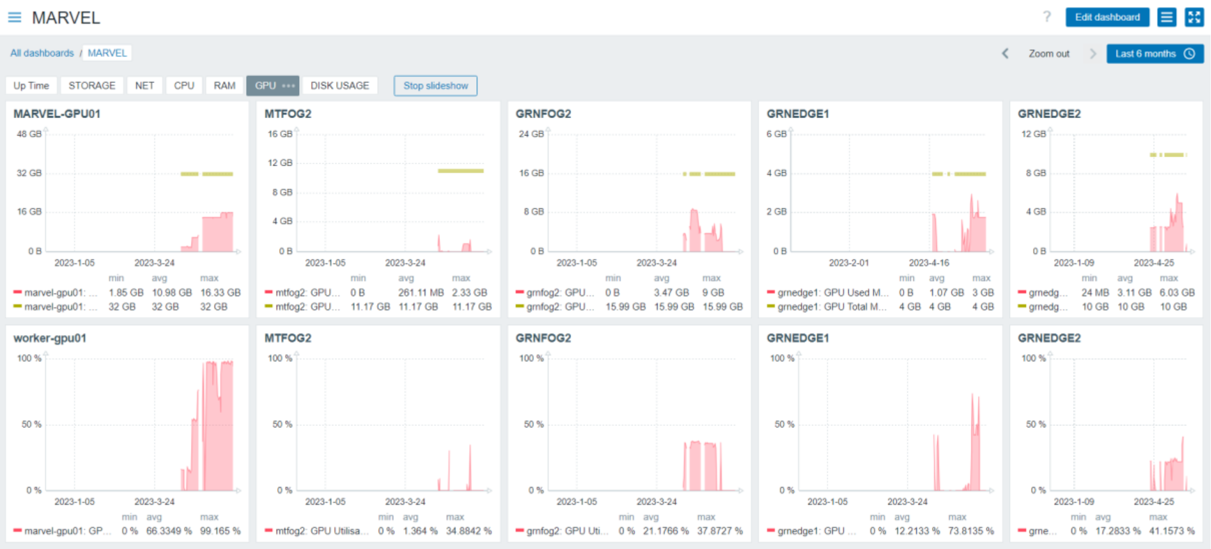
GRNFOG2 mqtt-kubernetes

```
> 2023-06-08 11:40:36 1686213636: New client connected from 10.244.11.194:59468 as c
> 2023-06-08 11:40:36 1686213636: New client connected from 10.244.11.194:59460 as 2
> 2023-06-08 11:40:36 1686213636: New client connected from 10.244.11.194:59448 as 5
> 2023-06-08 11:40:36 1686213636: New connection from 10.244.11.194:59468 on port 18
> 2023-06-08 11:40:36 1686213636: New connection from 10.244.11.194:59460 on port 18
> 2023-06-08 11:40:36 1686213636: New connection from 10.244.11.194:59448 on port 18
> 2023-06-08 11:40:36 1686213636: New client connected from 10.244.11.194:59442 as 0
> 2023-06-08 11:40:36 1686213636: New connection from 10.244.11.194:59442 on port 18
> 2023-06-08 11:40:36 1686213636: New client connected from 10.244.11.194:59426 as f
> 2023-06-08 11:40:35 1686213635: New connection from 10.244.11.194:59426 on port 18
> 2023-06-08 11:40:35 1686213635: New client connected from 10.244.11.194:59422 as a
```

Figure 22: MARVdash monitoring dashboard: MQTT broker status

- **MARVEL infrastructure monitoring tool**. During R2 integration activities, another dedicated tool was implemented and employed to further reinforce the capabilities for monitoring the MARVEL framework. This tool was based on the Zabbix[12] technology stack and was primarily focused on monitoring the infrastructure and its resources using GUIs that comprise dashboards, while it is also able to generate alerts and reports. This tool was used to provide a comprehensive presentation of each infrastructure node that is attached to the MARVEL Kubernetes cluster, including a live visualisation of the most significant metrics, i.e., CPU, RAM, GPU, DISK USAGE, Network speed (Figure 23). This functionality was particularly useful during stress tests that were carried out as part of the E2E integration testing activities.

[12] https://www.zabbix.com/



MARVEL-GPU01

	min	avg	max
🟥 marvel-gpu01: ...	1.85 GB	10.98 GB	16.33 GB
🟨 marvel-gpu01: ...	32 GB	32 GB	32 GB

MTFOG2

	min	avg	max
🟥 mtfog2: GPU...	0 B	261.11 MB	2.33 GB
🟨 mtfog2: GPU...	11.17 GB	11.17 GB	11.17 GB

GRNFOG2

	min	avg	max
🟥 grnfog2: GPU...	0 B	3.47 GB	9 GB
🟨 grnfog2: GPU...	15.99 GB	15.99 GB	15.99 GB

GRNEDGE1

	min	avg	max
🟥 grnedge1: GPU Used M...	0 B	1.07 GB	3 GB
🟨 grnedge1: GPU Total M...	4 GB	4 GB	4 GB

GRNEDGE2

	min	avg	max
🟥 grnedg...	24 MB	3.11 GB	6.03 GB
🟨 grnedg...	10 GB	10 GB	10 GB

worker-gpu01

	min	avg	max
🟥 marvel-gpu01: GP...	0 %	66.3349 %	99.165 %

MTFOG2

	min	avg	max
🟥 mtfog2: GPU Utilisa...	0 %	1.364 %	34.8842 %

GRNFOG2

	min	avg	max
🟥 grnfog2: GPU Uti...	0 %	21.1766 %	37.8727 %

GRNEDGE1

	min	avg	max
🟥 grnedge1: GPU ...	0 %	12.2133 %	73.8135 %

GRNEDGE2

	min	avg	max
🟥 grne...	0 %	17.2833 %	41.1573 %

Figure 23: MARVEL infrastructure monitoring tool based on Zabbix

3.8.4 Technical Validation Testing

This is the final and formal end-to-end integration test that was performed to validate the expected operation of a release of the entire MARVEL framework when applied in each use case that needs to be addressed.

At the end of the R2 integration activities in M30, end-to-end integration tests were planned for performing a technical validation of the MARVEL prototype operation in all addressed use cases. In this context, components were deployed at the foreseen infrastructure node according to the R2 use case requirements and foreseen system design specifications. An exhaustive list of all pairwise integration tests for each use case was prepared according to the component allocation to each use case and foreseen interactions between them. This list was used to document the result of each integration test, serving the purposes of technical validation. In the last month of R2 integration activities, **a two-day technical meeting was organised at Aarhus, Denmark with the exclusive purpose of carrying out the R2 validation tests**. The resulting R2 Technical Validation Test Report is attached in Appendix A.

4 Subsystems and Components integrated in R2

In this section, we provide technical information about the components that take part in one or more use cases for R2, grouped under the respective subsystem each belongs to. For each component, we provide a description, the technologies employed, and the role of that component in R2. Up-to-date information is provided reflecting the state of components as they were implemented in R2.

Table 3 below presents a comprehensive view of all the components that are integrated in R2 alongside with their main achievements and updates and the use cases where they are applied in the R2 phase.

Table 3: List of components integrated in R2 and main achievements / updates

Subsystem	Component	Partner	MVP Status	R1 Status	Main achievements / updates in R2 (after R1)	Applicable R2 use cases
Sensing and perception	Advanced MEMS microphones	IFAG	Partially integrated	Integrated	New interface	MT2, MT3, MT4, UNS1, UNS2
	AVDrone	UNS	N/A	Integrated	Rasberry Pi version 4 was implemented instead of Rasberry Pi version 3 as a computational device for processing audio signals due to malfunctioning caused due to continuous work for several months.	UNS1
	AV Registry	ITML	Not foreseen	Integrated	Updated hosted information on AV sources	ALL
Security, Privacy and data protection	EdgeSec VPN	FORTH	N/A	Integrated	Installed EdgeSec VPN in new nodes: worker2, worker3, worker4, worker-GPU, Zabbix server, grnedge2, grnedge3, grnedge4, grnfog2, unsedge3	ALL
	EdgeSec TEE	FORTH	N/A	Qualified and deployed, but not integrated/tested	• EdgeSec TEE is deployed on UNSEDGE1 (Intel NUC) and configured for the needs of UNS1, i.e., exchange data with the VideoAnony instance hosted on the same node over a REST protocol. • VideoAnony on UNSEDGE1 is appropriately configured to exchange data with EdgeSec TEE over a REST protocol. • Created a MARVdash YAML template with all related containers to deploy in UNSEdge1.	UNS1
	VideoAnony	FBK	Partially integrated	Integrated	no updates wrt R1 (except some minor fixing)	GRN1, GRN2, GRN3, GRN4, MT1, MT2, MT3, MT4, UNS1

	AudioAnony	FBK	N/A	Integrated	Updates: - deployment on 6 devices in MT and 2 devices in UNS - different voice conversion strategy - automatic reset of the device	MT2, MT3, MT4, UNS1, UNS2
	Voice Activity Detection (VAD) - devAIce	AUD	N/A	Integrated	No updates	MT2, MT3, MT4, UNS1, UNS2
Data Management toolkits	Data Fusion Bus (DFB)	ITML	Integrated	Integrated	- Updated fusion service for ViAD, AVAD and extended it to support SED, AT - Improved performance measurement and monitoring features - Updated ES-proxy service	ALL
	StreamHandler	INTRA	N/A	Integrated	- Updated Architecture and functionality (based on microservices) - New service introduced (DFB-SH)	GRN1, GRN2, GRN3, MT1, MT2, MT3, MT4, UNS1, UNS2
	DatAna	ATOS	Integrated	Integrated	- Deployment of new nodes in the use cases - Support for all use cases - Added data flows for the new inference models (YOLO-SED, RBAD, SELD, GPURegex) Support for versioning of data flows using the NiFi Registry	ALL
	Hierarchical Data Distribution (HDD)	CNR	N/A	Integrated	No updates.	ALL
Audio, visual and multimodal AI	CATFlow	GRN	Integrated	Integrated	Changes were done to the Configurator– which is responsible for configuring and deploying CATFlow, i.e., to accept parameter input on MARVdash for setting the MQTT host and port.	GRN2, GRN3, GRN4, MT1, MT4
	Text Anomaly Detection (TAD)	GRN	Not foreseen	Integrated	The TAD component was updated for R2 integration to be able to distinguish between detector errors which give anomalous speeds and real anomalous speeds. In addition, the speed per path taken by vehicle was taken into account.	GRN2, GRN3, GRN4
	Visual anomaly detection (ViAD)	AU	N/A	Integrated	A weekly-supervised visual anomaly detection method was developed. This method was integrated to the ViAD component in the MARVEL framework.	MT1
	Audio-Visual anomaly detection (AVAD)	AU	N/A	Integrated	New methodology was developed and tested on the new audio-visual anomaly detection dataset created by MARVEL. This method was integrated to the AVAD component in the MARVEL framework.	GRN3, MT2, MT3, MT4
	Visual crowd counting (VCC)	AU	N/A	Integrated	No updates.	MT1, UNS1

	Audio-Visual crowd counting (AVCC)	AU	Integrated	Integrated	No updates.	GRN4
	Automated audio captioning (AAC)	TAU	N/A	N/A	- AAC component was first released in R2 for MT2 use case	MT2
	Sound event detection (SED)	TAU	Integrated	Integrated	- SED component was applied in GRN2, GRN4, MT2, MT3 and MT4 use cases. - AI model training process extended with transfer learning-based process to increase the generalisation ability of the model	GRM2, GRN4, MT2, MT3, MT4
	Audio Tagging (AT)	TAU	Not foreseen	Integrated	- AT component was applied in MT2 and MT4. - AI model training process extended with transfer learning-based process to increase the generalisation ability of the model	GRN3, MT2, MT3
	Sound event localization and detection (SELD)	TAU	N/A	N/A	- SELD component was first released in R2 for UNS2 use case	UNS2
	YOLO-SED	AU-TAU	Not foreseen	Not foreseen	New component developed for R2	GRN1
	Rule-Based Anomaly Detection (RBAD)	AU	Not foreseen	Not foreseen	New component developed for R2	GRN2
Optimised E2F2C processing and deployment	GPURegex	FORTH	N/A	Qualified and deployed, but not integrated/tested	1) new image for NVIDIA GPU - 2) deployed on MTFOG2 - 3) integrated with MARVEL (receives input from AAC, sends output alerts to SmartViz, communication through DatAna) - 4) will be used in MT2	MT2
	DynHP	CNR	N/A	Integrated	- refactored the code - applied to AVCC and VCC model - Applied to MT1 data	AI Training (see Section 5.4.11)
	FedL	UNS	N/A	Integrated	- incorporated new model for audio-visual emotion recognition	AI Training (see Section 5.4.11)
	MARVdash	FORTH	Integrated	Integrated	• Installed Prometheus, Loki, Grafana and included them in the menu • Optimised the creation of services to put less effort to the user • Installed Kubernetes tools to all new nodes at all layers • Provided continued support for all the services deployed through MARVdash to Kubernetes cluster	ALL

HPC	HPC	PSNC	Integrated	Integrated	- extension of cloud resources to include BST region (in R1 only the DCW region was used) - providing VM equipped with GPUs to enable deployment of components that require constant access to GPU	ALL
	HPC management & orchestration	PSNC	Integrated	Integrated	- providing zabbix monitoring dedicated to MARVEL	ALL
User interactions and decision making toolkit	SmartViz	ZELUS	Integrated	Integrated	- applied for all 10 use cases - new support for GPURegex, RBAD, YOLO-SED, SELD, AAC, ViAD, fused results - MARVdash selvice to initialise and terminate a deployed component - new widgets and functionalities (sound map localisation, comparison etc.) - filtering options	ALL
	MARVEL Data Corpus as a service	STS	Integrated	Integrated	- Native installation of Hadoop Distributed system (HDFS) on all implicated/available nodes of Data Corpus. - Deployment of several REST APIS to facilitate the download capability for individual injected snippets. - Deployment of advanced filtering for the available injected datasets to the Corpus. - Deployment of an additional GUI for the Data Corpus in order to show the public available datasets.	Data Aggregation (see Section 5.4.12)

4.1 Sensing and perception Subsystem

4.1.1 MEMS microphone IM69D130

4.1.1.1 Overview

The XENSIV^{TM} IM69D130 MEMS microphone is the basis of the audio data acquisition. All evaluation Kits feature the latest state-of-the-art IM69D130 MEMS microphone. It is the newest MEMS microphone technology produced by IFAG for consumer electronics. It presents significative improvement in operating parameters, frequency response, performance and other acoustic features with respect to the state of the art (i.e., IFAG IM69D120).

4.1.1.2 Internal Operation & Technologies

Infineon's Dual Backplate MEMS technology is based on a miniaturised symmetrical microphone design, similar to studio condenser microphones, and results in high linearity of the output signal within a dynamic range of 105 dB. The microphone distortion does not exceed 1 percent even at sound pressure levels of 128 dBSPL. The flat frequency response (28 Hz low-frequency roll-off) and tight manufacturing tolerance result in close phase matching of the microphones, which is important for multi-microphone (array) applications.

The digital microphone ASIC contains an extremely low-noise preamplifier and a high-performance sigma-delta ADC. Different power modes can be selected in order to suit specific current consumption requirements. Each IM69D130 microphone is trimmed with an advanced IFAG calibration algorithm, resulting in small sensitivity tolerances (±1 dB). The phase response is tightly matched (± 2°) between microphones, in order to support beamforming applications.

4.1.1.3 Role in R2 and associated use cases

For the MT use cases, IFAG provided the devices to acquire the audio. In particular, Audiohub Nano and Audiohub 8-microphone versions have been distributed to the pilot owner. Together with FBK and AUD, the boards have been integrated in the anonymisation pipeline, to provide audio data removing sensible information from the public. This data can be further used by the rest of the partners to work on the sound event detection and localisation (i.e., by the SED component developed by TAU).

4.1.2 AVDrone

4.1.2.1 Overview

AVDrone represents an Edge-based data capturing and streaming component, that is used for audio and video monitoring of public events. The component is designed and tested according to the needs of UNS Use Case 1. The goal of the component is to enable target users to detect the number of people at a public event, providing a faster response in the case of reaching a user-defined threshold, hence preventing overcrowding, which could lead to dangerous situations, and helping to preserve public safety. The innovative setup includes video capturing from the drone and audio capturing from ground-based MEMS microphone, which serves as a supporting hardware for determining human presence by detecting voice activity, helping to localise the event. Data is stored locally but also streamed in real-time to an external IP address using Wi-Fi connection.

4.1.2.2 Internal Operation & Technologies

The AVDrone component encompasses two distinct groups of data capturing devices. Firstly, it incorporates a DJI M600 Pro[13] drone equipped with a GoPro camera to facilitate aerial video recording. Secondly, an IFAG MEMS microphone is deployed on the ground to facilitate additional data collection, specifically supporting voice activity detection. For data processing, an Intel NUC Mini PC serves as the computational device on the drone platform, while a Raspberry Pi v4 board is employed on the ground to process audio signals. The Intel NUC operates the VideoAnony software, enabling anonymisation of video data, while the Raspberry Pi board hosts AudioAnony, VAD, and DataAna Edge (MQTT).

4.1.2.3 Role in R2 and associated use cases

The AVDrone component is implemented within UNS1 use case – Drone experiment. The role of the component in R2 is to support real-time audio-visual intelligence by providing hardware with supporting software for data recording and streaming within UNS1 use case. The distinct features of AVDrone, with respect to the AV source produced, are bird-eye view video as well as the possibility of varying and controllable field of view enabled by the drone mobility and flight control. The component will not be implemented within other use cases.

Within R2, the component configuration will include only one microphone board at the ground, to support real-time inspection of the audio at the user request. For future implementations, deployment and synchronised streaming from several ground-based microphone boards is planned to enable SELD.

[13] https://www.dji.com/gr/matrice600-pro

4.1.3 AV Registry

4.1.3.1 Overview

This component was not originally foreseen as part of the MARVEL framework. Following the analysis of system integration requirements for R1, it was acknowledged that certain necessary functionalities related to the reception, analysis, and monitoring of live AV data streams were not being supported. In order to overcome this gap, it was decided that the AV Registry component should be developed by ITML and added to the MARVEL framework.

The purpose of the AV Registry is to host metadata information related to AV sources that describe their characteristics (e.g., name, type, URL, encoding, video resolution, video framerate, audio channels, etc) and to expose this information to other MARVEL components during runtime.

4.1.3.2 Internal Operation & Technologies

The AV Registry hosts a data store of documents that describe the AV sources that have been added to a deployment of the MARVEL framework. The AV sources can be cameras (possibly with built-in microphones) and microphones, but also instances of the MARVEL VideoAnony and AudioAnony components that provide anonymised AV data streams and act as AV sources as far as other MARVEL components are concerned.

Each AV source is described in a JSON document. A relevant data model from the Smart Data Models (SDM) [14] initiative was selected for describing all MARVEL AV sources to comply with industry standards. Specifically, the SDM "Camera"[15] data model was selected. This data model was extended by adding more fields to account for MARVEL-specific aspects of AV sources (Section 5.3.1). However, this extension did not break the compliance with the SDM standard, as the original fields were preserved and used for the purposes of MARVEL, wherever possible. The mandatory fields specified by SDM are considered mandatory in the revised data model that was created for MARVEL. In addition, the "Camera" data model was revised to include a specification of additional mandatory fields for the context of MARVEL in order to meet integration and other requirements. A complete specification of the revised "Camera" data model was produced in a mark-down (.md) format and was regularly updated during the R1 and R2 integration activities. The data model specification was hosted on the project's GitLab repository for version control. The complete Camera data model specification can be found in Appendix B of this report.

The AV Registry exposes two REST API calls for retrieving the complete set of hosted AV source documents and for retrieving a specific AV source document by providing the AV source id (Section 5.2.2).

4.1.3.3 Role in R2 and associated use cases

The AV Registry was initially developed for the integration needs of the R1 use cases and was applied in all ten R2 use cases. Each AV Registry instance was foreseen to contain information on the AV sources that are available for a particular use case. Due to this close coupling with use cases, different AV Registry instances were deployed for each pilot (GRN, MT, UNS) at the respective pilot's infrastructure Fog nodes. Each AV Registry instance hosted JSON documents that corresponded to the AV sources of the respective pilot. In each use case, AV

[14] https://github.com/smart-data-models

[15] https://github.com/smart-data-models/dataModel.Device/tree/master/Camera

Registry served the needs of all components deployed on all layers and infrastructure nodes that required metadata information of available AV sources. In the case of AI components and StreamHandler, during initialisation, they were required to access the AV Registry to receive the information of the AV sources they would need to connect to and process. In the case of SmartViz, it could query the information for a particular AV source from AV Registry upon demand, when a user would want to monitor the live stream of a particular AV source.

During R2 integration activities, the JSON documents hosted by the AV Registry instances were updated to incorporate revised information about all the active AV sources used across the ten use cases, including multiple AV sources that were introduced in R2. Furthermore, small revisions were also made to the data model describing AV sources.

4.2 Security, Privacy and data protection Subsystem

4.2.1 VideoAnony

4.2.1.1 Overview

The goal of VideoAnony is to anonymise the detected faces and car plates from raw video feeds coming from the CCTV cameras from each pilot site. The anonymisation is performed via image redaction methods, starting from classic image processing techniques, such as blurring, towards the more advanced GAN-based face-swapping techniques, which are under development within the MARVEL project. The component receives the incoming raw video stream either via RTSP or direct cable access and processes it with the face/car plate detection and anonymisation modules and finally streams the anonymised videos via a customised RTSP server (see Section 5.2.3 - AV streaming). The component will be deployed in a Fog machine, or an Edge device given sufficient on-board processing power to allow real-time processing.

4.2.1.2 Internal Operation & Technologies

The current deployable version of VideoAnony employs the YOLOv5 detector for face and car plate detection which is finetuned with related public datasets and pilot-provided annotations. Once the regions of interest are detected, the component then blurs them. On top of that, we are also developing a lighter version of the advanced GAN-based face-swapping model based on state-of-the-art methods. In the early phase, we particularly addressed the challenges of pose preservation and varying size of the detected faces from CCTV videos, while in the current phase, we are focusing on reducing the computational complexity of the model.

4.2.1.3 Role in R2 and associated use cases

The role of this component is to anonymise video streams from pilot sites that will expose car plates and faces of citizens. The component was used in the MVP for providing anonymised videos into the Data Corpus in an offline manner. VideoAnony is implemented in all use cases, over each of the AV streams with presence of video modality. The component is deployed on devices that either operate at the Fog layer (MT) or Edge layer (GRN and UNS use cases), depending on the specifics of the use case. In GRN the component is deployed on GRNEDGE1, GRNEDGE2. In MT it is deployed on MTFOG1 (outside the Kubernetes cluster) and on a Jetson device for MT3. In UNS it is deployed on UNSEDGE1.

4.2.2 AudioAnony

4.2.2.1 Overview

The goal of AudioAnony is to manipulate speech segments captured by the microphones in order to mask the speaker identity and preserve the privacy of the citizens. The component reads an audio stream, processes it and publishes the anonymised audio stream via an RSTP server (see Section 5.2.3 - AV streaming for more details about the RSTP server).

The AudioAnony component is coupled with the Voice Activity Detection (VAD) component to limit the anonymisation process to speech segments only. Given the very limited computational requirements of the current version, it will be deployed on Edge devices, getting the audio stream directly from the microphones.

4.2.2.2 Internal Operation & Technologies

The first version of the tool employs McAdams coefficients, shifting the poles on the speech LPC representation. Segments classified as speech by the VAD component are converted into their LPC representation. Poles are shifted using a random coefficient and then time domain signals are reconstructed. The final implementation of the AudioAnony component deployed in the inference pipeline employs a set of signal-processing based transformations of the audio signal to mask the speaker identity. The code uses the "Parselmouth" Python library for the Praat software[16]. The transformations are those described in ContentVec[17]. The AudioAnony component works in combination with VAD and is deployed on the Raspberry PI collecting the audio signals from the microphones so that audio is anonymised at the source. The audio stream captured by the microphone is imported using the ALSA API and processed in segments of predefined length (default=250ms). Note that besides the signal-processing based component for the inference pipeline, a voice conversion audio anonymisation tool has been developed for the offline processing of recorded data.

4.2.2.3 Role in R2 and associated use cases

The goal of the component is to anonymise audio streams at the source to comply with data processing restrictions introduced by the data providers, mostly for privacy issues. The component is therefore used in MT2, MT3, MT4, UNS1, and UNS2. GRN is not expecting any relevant speech signal, so AudioAnony is not deployed there. In MT the component is deployed on a Raspberry Pi device, on which the microphones are mounted. The devices are installed on lamp poles near the cameras selected for the use cases. In UNS the component is deployed as a docker container via MARVDash and reads a multi-channel audio stream made available via an RTSP server. The component publishes 1 single anonymised stream to an RTSP server for further processing.

4.2.3 Voice Activity Detection (VAD) – devAIce

4.2.3.1 Overview

devAIce is a software development kit wrapping AUD's intelligent audio analytics modules. devAIce contains various modules that can be used for different use cases and in different deployments environments, from powerful computing nodes to edge devices. One of these

[16] https://parselmouth.readthedocs.io/en/stable/

[17] https://github.com/auspicious3000/contentvec

modules is the Voice Activity Detection (VAD) module, which represents one of the core components in the MARVEL audio anonymisation pipeline, allowing the detection of sensitive audio data in noisy conditions which will be anonymised in the next stages of the pipeline.

4.2.3.2 Internal Operation & Technologies

The current VAD model in the SDK has been subject to multiple upgrades and improvements, in order to be more suitable for the MARVEL use cases. One of the core upgrades is retraining according to a novel state-of-the-art architecture. This architecture is inspired by a research paper published by Lee et al.[18], where a new approach is adopted. This new approach introduces the use of Convolutional Layers as an attention mechanism on top of the Recurrent Neural Networks (RNN) layers, which is a Long-Short Term Memory (LSTM) layer in our case. Moreover, this attention mechanism focuses not only on the temporal domain like the most commonly used Artificial Neural Network (ANN) based attention modules, but also on the frequential domain, which helps to achieve better modelling of the dependencies between the consecutive frames within an audio signal, by focusing on the parts that contain the most relevant information for Voice Activity, and this on both time and frequency domain. This concept is called Dual Attention. Moreover, this VAD module can now also detect music as well as voice activity. Further details on the attention module, the overall architecture as well as the most recent developments can be found in D4.1[19] and D4.4[20].

4.2.3.3 Role in R2 and associated use cases

As stated previously, devAIce VAD module makes up, together with AudioAnony the audio anonymisation pipeline. The VAD module, once audio data is consumed in real-time, will analyse the ingested audio streams, detect speech data in it if existing which will be anonymised right away with AudioAnony. The anonymised audio streams are later forwarded to the rest of the MARVEL AI components for further analysis. Furthermore, the VAD boundaries as well as the music boundaries, are forwarded through an MQTT broker to be stored in the MARVEL elastic search databases which will be visualised in later stages by the SmartViz.

This audio anonymisation pipeline is used in the MT and UNS use cases and was deployed on the corresponding edge devices. It was recently upgraded to support the anonymisation of 8-channels recording, using the recent version of the MEMS microphone for audio ingestion.

4.2.4 EdgeSec VPN

4.2.4.1 Overview

By implementing EdgeSec VPN, each physical or virtual host running the EdgeSec VPN software will join a peer-to-peer VPN network, enabling comprehensive encryption of network communication throughout the entire E2F2C architecture. EdgeSec VPN primarily aims to secure network traffic by encrypting it, safeguarding against unauthorised access, preserving the confidentiality and integrity of data during transit, and preventing any unknown or potentially malicious entities from gaining access to the associated host.

[18] Lee, J., Jung, Y. and Kim, H., 2020. “Dual Attention in Time and Frequency Domain for Voice Activity Detection”. INTERSPEECH 2020, (pp. 3670-3674).

[19] “D4.1: Optimal audio-visual capturing, analysis and voice anonymisation – initial version,” Project MARVEL, 2021. https://doi.org/10.5281/zenodo.5833277.

[20] “D4.4: Optimal audio-visual capturing, analysis and voice anonymization,” Project MARVEL, 2022. https://doi.org/10.5281/zenodo.7541704.

4.2.4.2 Internal Operation & Technologies

EdgeSec is a suite of components and the component that is referred as EdgeSec-VPN is the software n2n[21] from ntop. The n2n is a lightweight VPN, designed to create virtual networks that bypass intermediate firewalls with ease. To utilise n2n, two essential elements are required. The first element is a supernode responsible for enabling Edge nodes to announce and discover other nodes. This supernode needs to have a publicly accessible port on the internet. The second element comprises the Edge nodes that become part of the virtual network.

n2n seamlessly handles the encryption and decryption of data during host-to-host communication. Users of this component do not require any specific interface or connection, as these operations occur transparently. Upon deployment, each host is assigned a unique private IP address, and all communication between this host and other hosts within the E2F2C architecture is tunnelled directly through this private IP address.

The n2n supernode is currently installed on a virtual machine deployed on PSNC's OpenStack infrastructure.

4.2.4.3 Role in R2 and associated use cases

EdgeSec-VPN plays a crucial role in integrating each host within the E2F2C framework into the Kubernetes cluster, thereby enabling seamless accessibility through MARVdash. These hosts are machines owned by the MARVEL pilots that need to become part of the MARVEL E2F2C cluster as worker nodes. Each joined host becomes visible to MARVdash and becomes available as target deployment environment for the MARVEL components, through the automated deployment method offered by MARVdash. MARVEL E2F2C architecture exhibits a complex topology involving network address translation (NAT), as illustrated in Figure 24.

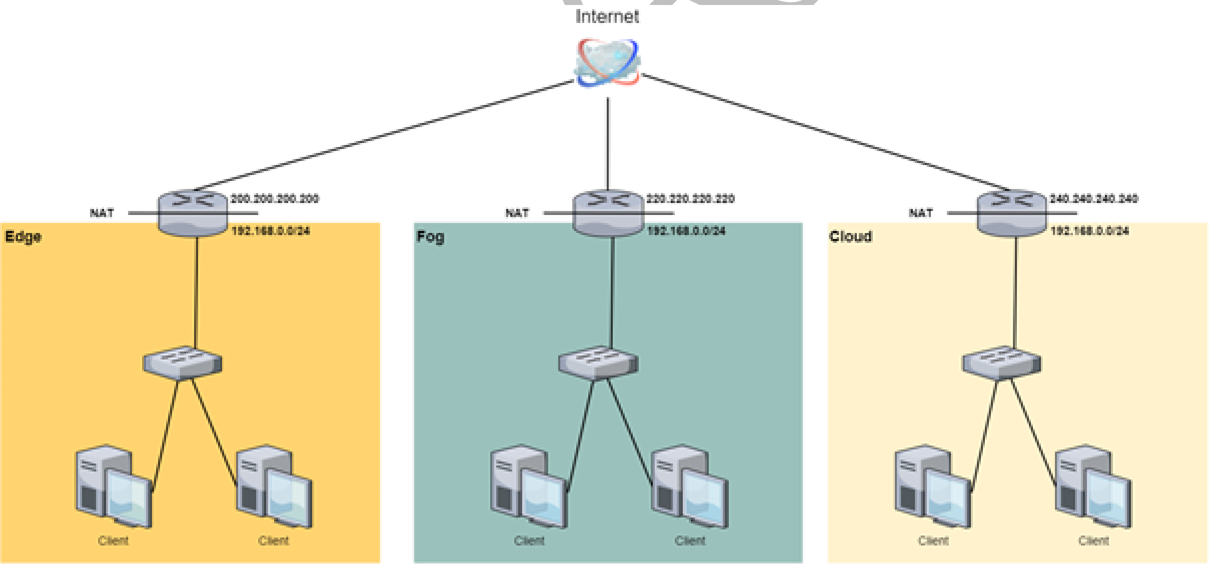


Figure 24: MARVEL E2F2C Network Architecture

By installing EdgeSec-VPN, participating hosts in the Kubernetes cluster can effectively traverse NAT and firewalls, making them reachable as if they were part of the same network (Figure 25). As a result, firewalls no longer impede direct IP-level communication.

[21] https://www.ntop.org/products/n2n/

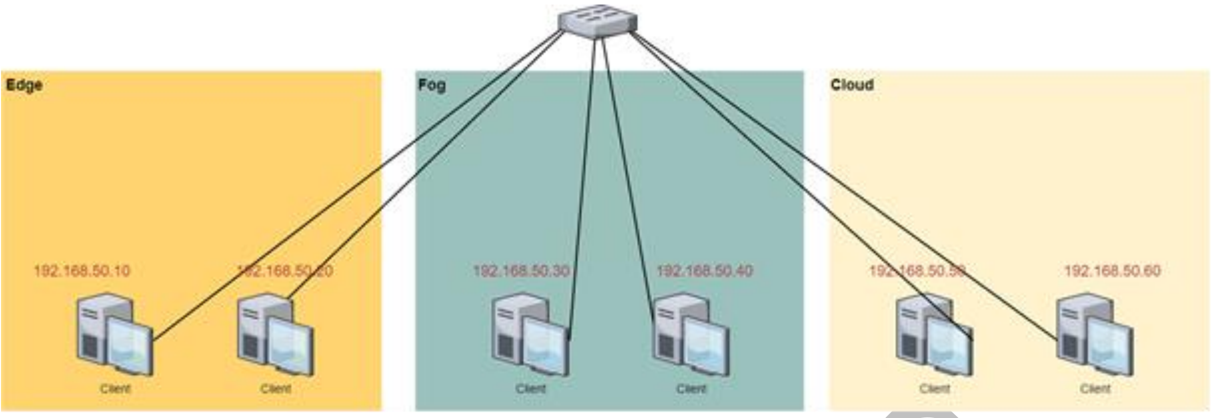


Figure 25: MARVEL E2F2C Architecture with VPN

For further details please refer to D4.5[22].

4.2.5 EdgeSec Trusted Execution Environment (TEE)

4.2.5.1 Overview

EdgeSec Trusted Execution Environment (TEE) aims to provide secure and confidential execution for sensitive data processing applications and components within MARVEL. It leverages Intel Software Guard Extensions (SGX) technology, which is supported by a subset of Intel processors. To ensure secure execution even within containers, EdgeSec TEE is integrated with Secure Container Environment (SCONE), a software platform designed to protect the data and code of applications running in Linux containers.

Secure containers are necessary when application or data owners lack trust in the container hosts. For example, according to the threat model of Intel SGX and TEEs in general, there is no requirement to trust the OS, hypervisors, or BIOS. SCONE assumes the presence of a powerful adversary with superuser access to the system, including access to the physical hardware and control over the entire software stack, such as the container engine and the OS kernel.

4.2.5.2 Internal Operation & Technologies

EdgeSec TEE is built upon the Intel SGX technology, which allows the protection of specific code and data from unauthorised access and modifications. With Intel SGX, application developers can divide their applications into two parts: the sensitive portion that requires integrity and data protection, and the remaining non-sensitive part. Intel SGX achieves trusted execution by isolating the sensitive code within encrypted memory regions called memory enclaves. Additionally, Intel SGX provides remote attestation, a security feature that verifies the integrity of the enclave before transmitting data to it. To extend trusted execution to containerised environments, EdgeSec TEE utilises SCONE, a secure container technology for Docker that leverages the trusted execution capabilities of Intel processors with SGX support. SCONE safeguards container applications from external attacks.

To utilise EdgeSec TEE, Docker and an Intel SGX-enabled processor are required. The EdgeSec TEE (v0) can be obtained from the MARVEL image registry.

[22] "D4.5 - Security assurance and acceleration in E2F2C framework – final version," Project MARVEL, 2023. https://doi.org/10.5281/zenodo.8147058.

4.2.5.3 Role in R2 and associated use cases

In R2, the EdgeSec TEE is combined with VideoAnony since it handles sensitive data such as URL, username and password of camera feeds. At its initial implementation, VideoAnony accepts as a parameter a string with all the aforementioned sensitive data. This sensitive string is stored as a Kubernetes Secret so it is protected in the YAML template that is stored in MARVdash. However, if an unauthorised user gets access to the host machine that VideoAnony is finally deployed, then he is able to see that sensitive string simply by issuing the Linux top command. Additionally, if the unauthorised user gets access to the container itself then he is also able to view the information provided by the Kubernetes Secret because they are stored in environmental variables accessible via the printenv command.

To address the above issues, we needed to take several actions. Firstly, to tackle the issue that the information in the sensitive string is visible via the htop command, we needed to change the way VideoAnony receives this information. To that end, we created a simple HTTP service based on python flask that its endpoints provide the sensitive information to VideoAnony. This interaction removes the sensitive information from the htop command. Additionally, the Kubernetes Secrets were not used, so the unauthorised user cannot access this information via the environmental variables.

The EdgeSec TEE concept is applied to the HTTP service mentioned. With the process of sconification, the container that runs the HTTP service is executed at protected private regions of memory, called enclaves. Even if an authorised user manages to get access to the container itself, he will not be able to see the content of the Python code because it will be scrambled.

The host machine that meets the hardware requirements for the use of EdgeSec TEE is an Intel NUC used in the use case UNS1 “Drone experiment”.

For further details please refer to D4.5 *‘Security assurance and acceleration in E2F2C framework – final version’*.

4.3 Data Management and distribution Subsystem

4.3.1 DatAna

4.3.1.1 Overview

DatAna is a component that allows performing data ingestion, transformation, enrichment and movement across the computing continuum. DatAna enables users to define data flows in a graphical and easy manner, with almost zero code and a soft learning curve. DatAna may output their results and ingest data to/from a high variety of systems, many of them already built-in in NiFi, which makes it ideal as middleware for data handling and transformation. It works both with static data (i.e., from files in different formats, databases) or data streams (i.e., MQTT or Apache Kafka topics).

4.3.1.2 Internal Operation & Technologies

DatAna is an Apache NiFi-based component that can be deployed in the different layers of the system. The underlying technologies of DatAna (mainly Apache NiFi) provide non-functional properties to the DMP such as data provenance tracking, scalability, high throughput, low latency, backpressure (managing internal queues to avoid being overrun by data), and the possibility of changing the data flows at runtime, among others.

The main components of DatAna used in MARVEL are the following:

- Apache NiFi: DatAna relies on the functionality of the Apache NiFi ecosystem (version 1.15.3). In MARVEL, Apache NiFi has been prepared to work within the MARVEL Kubernetes cluster as a set of services in MARVdash located in each of the layers of the architecture (edge, fog and cloud) for each of the pilots. To do so, yaml configuration files have been provided and adapted for the project needs.
- Apache NiFi Registry: DatAna uses a NiFi Registry located at the MARVEL cloud, deployed in Kubernetes using MARVdash, to keep track of the different versions of the data flows designed for all the use cases. The use of the registry avoids code and data flow design repetition. This is very handy in MARVEL, as many of the data flows designed to handle the different ML inference model outputs have several common blocks. Moreover, the inference components run in different infrastructures and layers for the same or different use cases (e.g., CATFlow runs for MT and GRN use cases). Therefore, the use of the Registry allows easy deployment of data flows, avoiding the duplication of code and thus minimising errors.
- MQTT: MQTT brokers (Mosquitto MQTT) serve as communication hubs between the inference models and DatAna. This allows the decoupling of the functionality of the ML inference models and the data management platform, therefore isolating their functionality and facilitating the integration of the different modules.

For secure communication between the different layers of the computing continuum, DatAna uses NiFi TLS security. In this way, the different NiFi services in Kubernetes located in the infrastructure provided for the pilots (edge, fog and cloud) are connected via the NiFi Site-to-Site (S2S) protocol in a secured manner, allowing the creation of NiFi topologies to communicate and move data throughout the layers adapted for the use cases.

4.3.1.3 Role in R2 and associated use cases

The main functionality provided by DatAna in the MARVEL project is acting as a central bus for the inference pipelines data outputs processing and transformation for all the use cases. This was the case since the MVP, where DatAna was used only in the cloud, but also for R1 with the main building blocks (except the NiFi Registry) already in place.

DatAna is an integral part of the data inference pipeline for all R1 and R2 use cases. The use of DatAna for the R2 release is therefore the following:

- An inference model, regardless of where it is deployed (edge, fog or cloud), during runtime provides its output as a stream of messages to the nearest MQTT broker, typically deployed in the same layer infrastructure, using a designated topic for each inference model (e.g., the SED model output their messages to a topic in MQTT called “sed”).
- The nearest NiFi of DatAna subscribes to the MQTT topics expected for the use cases and receives the messages.
- Depending on the nature and format of the messages provided by the inference model, DatAna processes those messages and transforms them to comply with the specific data models.

  - The agreed data models in MARVEL are based on and extend the Smart Data Models [23] for Media Events, Alerts, or Anomalies defined for R1 and kept in R2 (see section 5.3.2), thus making the inference data stored in MARVEL compliant with ad-hoc standards and therefore easy to interoperate with.
  - DatAna relies on some key attributes to ensure the traceability of the data for the specific use cases, such as the usage of absolute timestamps, or pointers to the source streams (cameras or microphones), among others.
- Once transformed to the agreed data models, if the inference occurs at the edge or at the fog layers, DatAna reroutes them to the cloud instance.
- Once in the cloud, DatAna forward the messages to specific agreed Kafka topics (e.g., for the messages coming from the SED model to a topic called "sed" in Kafka) of the DFB for further fusion, processing and storage.

In particular, in MARVEL, DatAna provides the following topologies for data management, as depicted in Figure 26:

- One instance of DatAna Cloud, consisting of the NiFi Registry, a common (master) NiFi and a Mosquitto MQTT deployed in the MARVEL Cloud via MARVdash. The NiFi in the cloud is connected via the NiFi S2S protocol to the rest of the NiFis residing at the Fog layers of the use cases.
- Instances of DatAna Fog, at least one per pilot, consisting of NiFi instances deployed via MARVdash in the different Fog servers of MT, GRN, and UNS, along with the corresponding MQTT brokers.
- Several instances of DatAna Edge, as required by the pilots and connected to their respective DatAna at the Fog. This will be at least MQTT brokers and in some cases NiFi instances deployed at the Edge.
  - It is important to point out that in the second iteration a decision was made to not use Apache MiNiFi at the edge. MiNiFi was initially foreseen as the DatAna deployment in edge devices (except for the PCs located at the edge in the GRN use cases), due to its lower footprint compared to NiFi. However, to avoid a negative impact on those devices, already supporting the execution of several inference models, anonymisation, or AV stream handling, the use of MQTT in combination with Apache NiFi at the fog has been proposed.
  - Different strategies may be used to ensure this communication, depending on the availability of the edge device in the Kubernetes cluster, etc. In the usual case, the inference models send their outputs to the MQTT at the edge and the nearest DatAna (NiFi) at the fog subscribe to those MQTT queues.
  - As a result, the messages arrive to the NiFi at the fog layer where they are handled by DatAna.

All this process is done in real time in a stream fashion due to the ability of NiFi of processing multiple concurrent data flows. DatAna topologies depicting the steps explained above for all covering the DatAna infrastructure released for R2 can be seen in Figure 26.

[23] https://smartdatamodels.org/

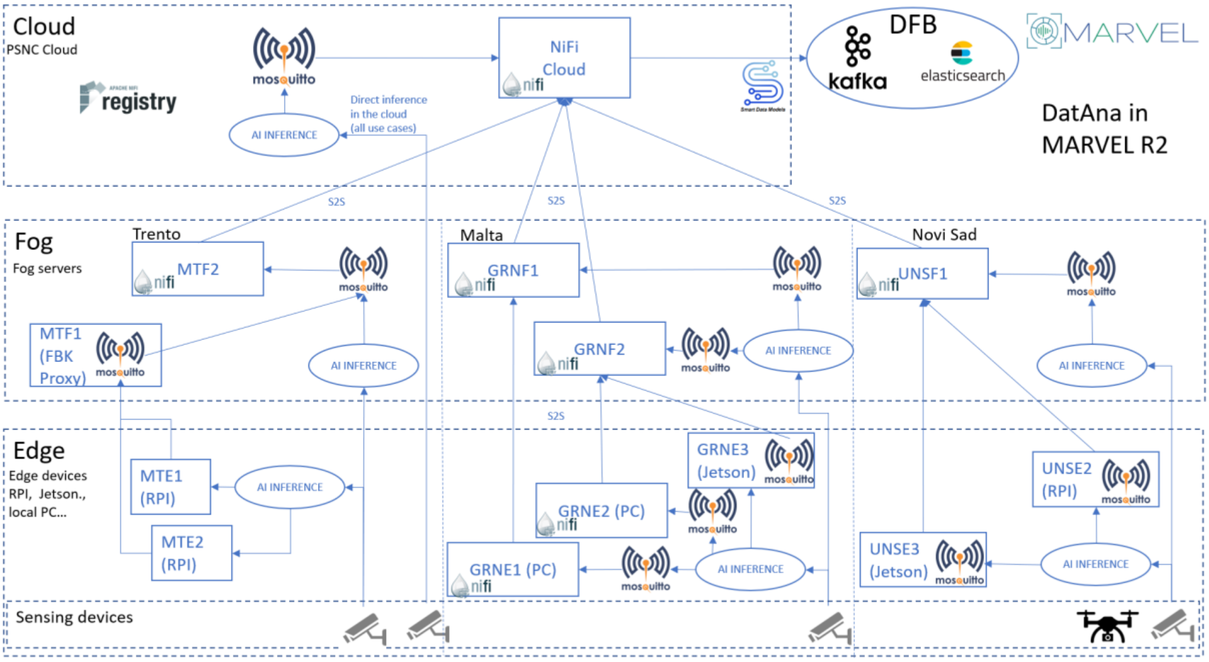


Figure 26: DatAna topologies for R2

Besides the central role in the inference pipelines, DatAna could also enable the possibility to perform other processes, such as enriching the data, examining thresholds based on the data received, or receiving control signals from different components. These possibilities have not been carried out in MARVEL but are extensibility points for future enhancements.

4.3.2 Data Fusion Bus

4.3.2.1 Overview

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

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

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

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

Within MARVEL, the DFB has a pivotal role in the Data Management Platform (DMP) and in the inference pipeline. Its main objectives are:

- Aggregate streamed inference results that originate from MARVEL AI components and are relayed by DatAna and expose them in real time to SmartViz, StreamHandler and Data Corpus.
- Permanently store the incoming inference results and make them accessible to SmartViz via a query system.
- Fuse incoming inference results from selected AI components (AVAD, ViAD, SED, AT) in real time and expose them to SmartViz.
- Allow SmartViz users to verify the validity of inference results that are stored in the DFB.
- Monitor real-time data traffic and visualise stored inference result data.
- Serve as an interface of the MARVEL framework towards third-party tools (outside of the MARVEL framework) that may need to send data (e.g., external databases or third-party, non-integrated AI components) or to receive data (e.g., third-party data post-processing and data visualisation tools).

The DFB is presented in more detail in the WP2 deliverables, i.e., D2.2[24] and D2.4[25].

4.3.2.2 Internal Operation & Technologies

The DFB has been customised and adapted for MARVEL with added services to meet the expected objectives. The key modules of DFB are:

a. Apache Kafka, an open-source framework for stream processing based on the publish-subscribe messaging pattern.
b. Elasticsearch, a distributed, multitenant-capable, data repository and search engine.
c. ES-Connector service, responsible for ingesting the messages received at the Kafka broker and permanently storing them on Elasticsearch
d. ES-Proxy, a REST API service that is accessed by SmartViz for performing data queries in the Elasticsearch repository, customised for the needs of MARVEL.
e. DFB Core & UI, implementation of a REST API and a client GUI, respectively, for management and monitoring of the DFB components.
f. Visualisation tool based on Grafana for monitoring the status, health and performance of the Kafka cluster.
g. Visualisation tool based on Kibana for inspecting, querying and visualising the data stored in the Elasticsearch repository
h. Inference result fusion service

The main inbound interface for DFB is Kafka's messaging system, which is based on the publish-subscribe pattern. More specifically, any component producing text-based data can connect to the DFB and publish its data to a specific, predefined topic.

Regarding outbound interfaces, in the general case, any data-consuming component can subscribe to a topic and receive instant updates on published data.

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

[25] D2.4 - Management and distribution Toolkit – final version," Project MARVEL, 2023. https://doi.org/10.5281/zenodo.8147109.

Within the context of MARVEL, data collected from the DFB Kafka brokers is subsequently passed onto an Elasticsearch Logstash Kibana (ELK) stack for storage and further processing and visualisation. Although DFB offers its own graphical UI for the visualisation of aggregated data based on the Kibana platform, collected streamed and stored data are made available to MARVEL's Decision Making Toolkit (SmartViz).

DFB is typically foreseen to be deployed on the Cloud. It is designed and optimised for handling non-binary data that is streamed to the Kafka interfaces in real time.

Figure 27 illustrates the specific implementation of DFB that was performed for the needs of the MARVEL integrated framework, initially for R1 and revised for R2.

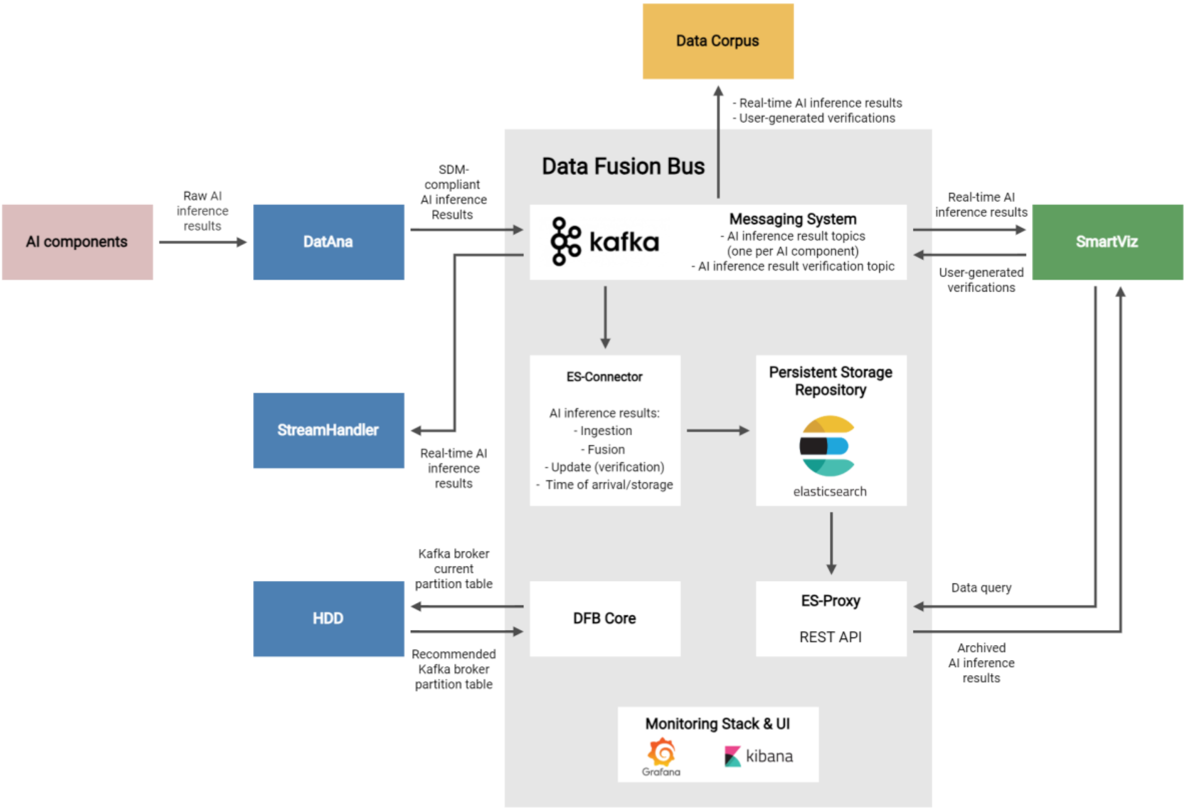

Data Corpus
- Real-time AI inference results
- User-generated verifications
Data Fusion Bus
AI components
Raw AI inference results
DatAna
SDM-compliant AI inference Results
kafka
Messaging System
- AI inference result topics (one per AI component)
- AI inference result verification topic
Real-time AI inference results
User-generated verifications
SmartViz
StreamHandler
Real-time AI inference results
ES-Connector
AI inference results:
- Ingestion
- Fusion
- Update (verification)
- Time of arrival/storage
Persistent Storage Repository
elasticsearch
Kafka broker current partition table
HDD
Recommended Kafka broker partition table
DFB Core
ES-Proxy
REST API
Data query
Archived AI inference results
Monitoring Stack & UI
Grafana
kibana

Figure 27: DFB internal architecture and interactions for the MARVEL R2

4.3.2.3 Role in R2 and associated use cases

In terms of the overall architecture of the MARVEL framework, the DFB is applied in all R2 use cases as it is considered a central hub for aggregating all inference results and relaying them to other components (SmartViz, DataCorpus). Therefore, the DFB is deployed in the Cloud at the PSNC HPC, and the same component instance is used for all use cases in order to collect the data and persistently store them. The DFB receives inference results from an instance of DatAna at the Cloud, which publishes inference results to Kafka topics of the DFB after transforming them into an SDM-compliant format.

Following the release of the MARVEL MVP and R1, the DFB was designed to accommodate the needs of the ten (10) use cases that were defined for implementation in the context of the final version of the MARVEL Integrated framework (R2). During R1 and R2 development and integration activities, the DFB functionalities were further elaborated, refined, and extended through the following activities:

- Configuration for receiving inference results in an SDM-compliant data models (MediaEvent, Alert and Anomaly) from DatAna.
- Configuration of an individual Kafka pub/sub-topic for each inference result type according to the AI producer component.
- Implementation of inference result verification process, i.e., accept inference result verification messages from SmartViz on a Kafka topic configured for this purpose and update corresponding entries in the Elastic Search (ES).
- Implementation and iterative revisions of a data fusion process for inference results that refer to single instants in time. Consecutive repetitive incoming results from the same AI producer (e.g., anomaly detection) that refer to a single time instant are identified and fused together to form a composite result that refers to a period of time. The start of the time period corresponds to the first occurrence of a repetitive event and the end of the time period corresponds to the last occurrence of that event.
- Revisions of the REST API implemented for performing ES queries (ES-proxy) based on updated requirements from SmartViz.
- Registering the time of arrival of each inference result at the Kafka broker and at the Elasticsearch data store for benchmarking purposes in dedicated time fields, present in the inference result data models.
- Implementation of the DFB monitoring stack.
- Connection with HDD to send Kafka topic partition data and receive optimised partition allocations recommended by HDD.
- Persistent volumes and enhanced retention policies implemented for storing DFB health and performance data.
- Fine-tuning and bug fixing following iterative participation in E2E integration testing activities.

4.3.3 StreamHandler

4.3.3.1 Overview

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

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

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

The key capabilities and features offered by the platform are:

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

- Security (encryption, authentication, authorisation

In this deliverable, the main focus will be given to the changes from an architectural perspective as well as the new components' interconnections introduced in R2.

The StreamHandler Platform consists of several components; however, the core functionality component was recently restructured. This restructure was made in order to enhance StreamHandler's efficiency and robustness. In Figure 28, can be seen how the core functionality has been divided into smaller services following the principles of the Microservice Architecture, the explanation of each service can be found in Table 4.

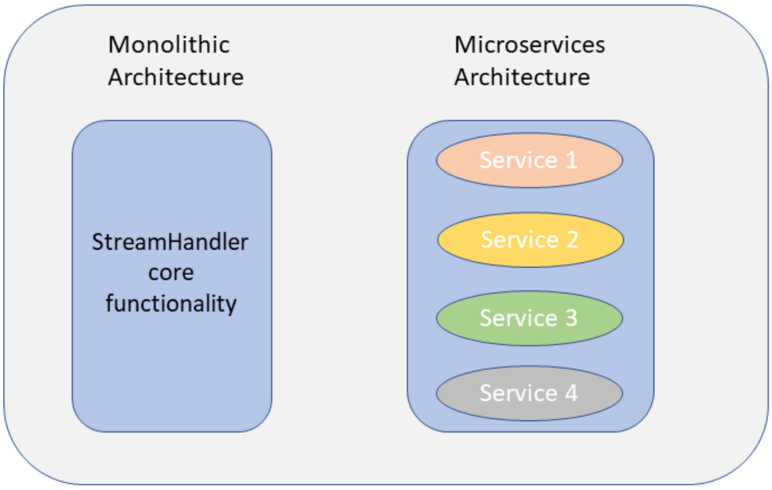


Figure 28. StreamHandler's Monolithic-Microservices Architectures

Table 4. StreamHandler's Service-Functionality

StreamHandler core functionality	Service Name	Functionality
	Service 1	Receive and store live streams in binary format.
	Service 2	Provide access to the stored audio-visual files to external components through a REST API.
	Service 3	Receive the information of an occurred event, retrieve from the storage component the audio-visual segments required and store in a binary format the event's audio-visual context.
	Service 4	Storage component; it is used for storing, retrieving and sharing the produced audio-visual files in binary format.

As presented in Figure 28, the core functionality of StreamHandler has been divided in four independent services. This guarantees that if a potential bottleneck occurs in the future due to a high processing load (e.g., higher resolution streams, more audio-visual sources), the platform can scale up by just replicating a parallel instance of this service. In addition to the StreamHandler's core reconstruction, a new interconnection with another component was introduced in R2. In R1, the StreamHandler was producing an audio-video file on-demand based on the stored audio-visual files, while in R2 the functionality of an event-based audio-video file production is added. This is achieved by interconnecting the StreamHandler with the DFB component described in Section 4.3.2.

4.3.3.2 Internal Operation & Technologies

The initial implementation of StreamHandler in R1 was based on Python and low-level system commands in a monolithic architecture with low scalability degree. The current implementation of the modules of StreamHandler in R2 is totally reconstructed compared to the one presented in R1. As presented in Section 4.3.3.1 of this document, the version deployed in R2 is based on microservice architecture providing more flexibility and stability over the monolithic approach in R1. The basic components of this newer version are:

- Java[26]
- SpringBoot[27]
- MinIO[28]
- Docker[29]
- REST API[30]
- Kafka[31]

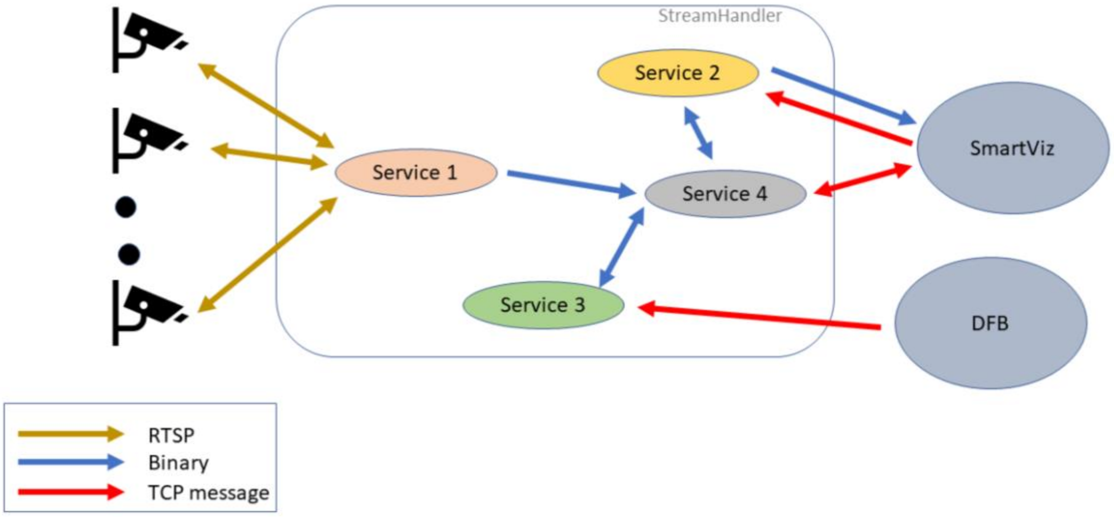


Figure 29. StreamHandler R2 implementation

As presented in Figure 29, the main tasks presented in Section 4.3.3.2 in D2.4 remain the same. In addition to them, one more task is added to the R2 version. This new task allows communication between the DFB component and StreamHandler. The main difference between R1 and R2 versions is the architecture used that decouples the relation between the several

[26] https://dev.java/

[27] https://spring.io/

[28] https://min.io/

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

[30] https://restfulapi.net/

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

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 audio-visual 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 audio-video creation, b) on-event audio-video creation and c) a list of segmented files. In the first case, the audio-visual segments from the storage component are retrieved and a new audio-video 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 audio-video source. The input parameter of this endpoint is sourceID.
- Service 3 is responsible for constructing audio-video 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 audio-video file.
- Service 4 is the storage component. This service is responsible for storing the audio-visual files produced by Service 1 and Service 3. In addition, the SmartViz component is directly retrieving from the storage component the audio-video file through a shared link from Service 2.

4.3.3.3 Role in R2 and associated use cases

StreamHandler was not implemented in the MVP, as its new functionalities were still under development at the time. In the context of R1, a different instance of StreamHandler is deployed for each pilot at the respective Fog node. The role of StreamHandler in R2 is similar to its role in R1. StreamHandler is deployed in the Fog layer in order to ensure that sensitive data will be stored as close as possible to the audio-video sources. This ensures that the aspect of data privacy is guaranteed while eliminating the possibility of data leak in unauthorised entities. As StreamHandler operates in the Fog layer, a per pilot deployment in the R2 phase is necessary. Currently, the new version of StreamHandler has been deployed in all three pilots, namely GRN Fog, MT Fog and UNS Fog servers.

4.3.4 Hierarchical Data Distribution (HDD)

4.3.4.1 Overview

HDD is a component which aims at facilitating the MARVEL distributed data delivery and access algorithmic schemes to guarantee delay and resource usage application requirements in internetworked environments. To do so, HDD has been tailored to suit the MARVEL data distribution technologies, and with a focus on providing optimisation recommendations on efficient data topic partitioning of Apache Kafka widely used by other MARVEL data distribution components (DFB). Even though Apache Kafka provides some out-of-the-box optimisations, it does not strictly define how each data topic shall be efficiently divided into partitions. The well-formulated fine-tuning that is needed to improve a Kafka cluster performance is still an open research problem and the main contribution of HDD. There are contrasting forces defining the number of partitions to be used, so choosing the "right" number of partitions per topic requires knowledge of many factors. Increasing the partition density (the number of partitions per broker) adds an overhead related to metadata operations and per partition request/response between the partition leader and its followers. Increasing the number of partitions in a cluster though will lead to increased parallelism of message consumption, which in turn improves the throughput of a Kafka cluster; however, the time required to replicate data across replica sets will also increase. The complete presentation of HDD's current design and operational parameters are presented in detail in MARVEL's D2.2.

4.3.4.2 Internal Operation & Technologies

The component first models the Apache Kafka topic partitioning process for given topics. Then, given the set of brokers and suitable application constraints and requirements (such as throughput, OS load, replication latency, and unavailability), formulates the optimisation problem of finding how many partitions are needed. It should be noted that this is a computationally in-tractable problem, as its form is that of an integer program. Therefore, the design of an exact, polynomial-time solution for any given instance of the problem is not feasible (unless "P=NP").

The running version of HDD incorporates two algorithms, both of which respect the hard and soft application constraints and requirements. The first algorithm targets at maximising the number of partitions needed for the given data topic allocation, while keeping the usage of the available brokers to a minimum extent. The second algorithm targets at maximising the number of partitions, while also maximising the usage of the available brokers. HDD is currently implemented in Octave SW suite[32]. This choice was made due to its open-source nature compared to commercial alternatives and its computational power when it comes to mathematical programming applications.

4.3.4.3 Role in R2 and associated use cases

In R1, HDD was focusing on performing some initial tests of updating offline selected topic partitions. During the initial end-to-end integration tests, HDD is fed with the required input and metrics from the DFB Kafka cluster. Then, HDD delivers a recommended partition table to be applied and tested during the final integration tests. Since it is burdensome to update and change the Kafka topic partitions when the system is live and operational, as this could compromise general system stability and Kafka brokers may be unable to serve producers/consumers, the periodical updates are intended to take place while the system services are not operational.

In R2, HDD is focusing on performing tests of updating selected topic partitions, with the final tests aimed at demonstrating the high scalability that the system can achieve, by highlighting the performance of selected parameters.

4.4 Audio, visual and multimodal AI Subsystem

4.4.1 CATFlow

4.4.1.1 Overview

CATFlow is a software asset developed by GRN, where the input is a video stream and the output is a list of traffic objects tracked over the camera field of view. The CATFlow component

[32] https://octave.org/

was updated to dynamically select the MQTT host and port from MARVdash for the Configurator.

4.4.1.2 Internal Operation & Technologies

CATFlow was already fully functional in the previous MARVEL release (MVP) and R1 integration. CATFlow was integrated in the MARVEL framework for R1. This effort mainly required containerisation of the CATFlow component. No changes were made to the internal operation of CATflow since the R1 integration. However, changes were done to the Configurator–which is responsible for configuring and deploying CATFlow, i.e.to accept parameter input on MARVdash for setting the MQTT host and port.

4.4.1.3 Role in R2 and associated use cases

CATFlow classifies vehicles in six different classes: car, bus, light goods vehicles, heavy goods vehicles, bicycle, and motorcycle. In addition, each object (e.g., vehicle or pedestrian) is tracked across the camera view and its trajectory is extracted and stored for visualisation or further processing. Pedestrians can also be detected and tracked. CATFlow is used in many of the MT and GRN use cases due to its ability to provide good performance across a variety of camera sources without the need to retrain the model. For each camera, the GRN developers only need to configure the vehicle/object entry and exit lines, along with location information pertinent to the camera.

- GRN1: The CATFlow detector model weights were used in the implementation of YOLO-SED. The detector weights allowed the use of the YOLO model without the need of any more data.
- GRN2: The CATFlow component was used to generate trajectories for the TAD component to detect anomalous trajectories in the data. The CATFlow data were also used by the RBAD component to detect anomalies such as jaywalking and buses not on time.
- GRN3: CATFlow is used to extract an estimate of the vehicle’s speed. This information is then passed onto the TAD component such that vehicles driving at anomalous low or high speed could be flagged.
- GRN4: CATFlow is used to detect and track the different types of vehicles and pedestrians. In addition, CATFlow outputs the trajectories of each vehicle as well as the entry and exit point within the camera field of view. All these data are then used in transport studies, e.g., to observe how the junction or road segment that is being monitored is used by pedestrians and vehicles.
- MT1: CATFlow was identified to be a good component to track pedestrians for the Piazza Fiera camera. Following some tests, CATFlow would also be able to provide the trajectories of these pedestrians. Another camera, deployed in this use case at Piazza Duomo, is characterised by a less than ideal angle and distance from the pedestrians. This may present a challenge for CATFlow to accurately detect pedestrians. Figure 30 shows CATFlow detecting pedestrians from the Piazza Fierra video stream.

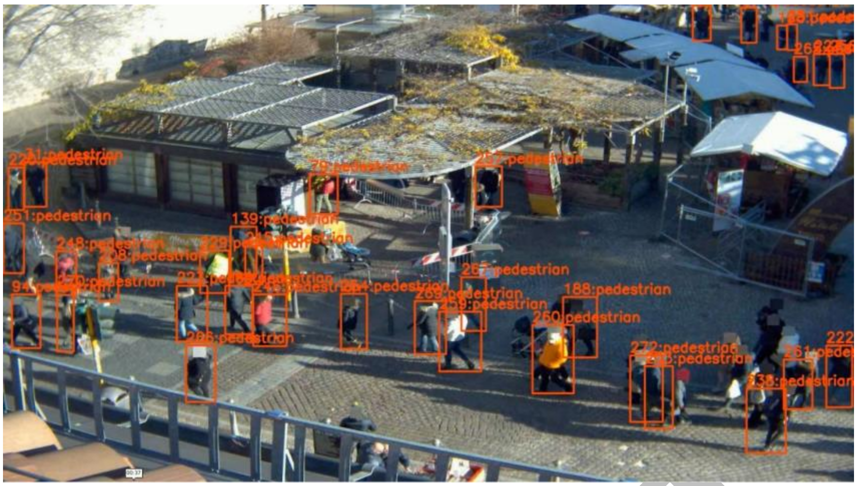


Figure 30: Screenshot of video processed by CATFlow

- MT4: The CATFlow component was used to analyse the areas by detecting the number of persons, number of cars, number of buses, number of bikes and the associated trajectories.

The CATFlow software used in every use case is the same, i.e., the same data model is used for the generated output and the user will have access to the fields that are required for the use case in hand. For each stream processed by CATFlow, a new configuration file needs to be created. This configuration contains information to set up the algorithm, i.e., the entry and exit lines on the video stream.

4.4.2 Text Anomaly Detection (TAD)

4.4.2.1 Overview

Text Anomaly Detection (TAD) is a component that automatically detects anomalous events in data, for example, anomalous vehicle velocities and trajectories. TAD takes as input the JSON messages outputted from CATFlow, and, after processing them, flags any anomalous behaviour. The TAD component was updated for R2 integration to be able to distinguish between detector errors which give anomalous speeds and real anomalous speeds. In addition, the speed per path taken by vehicle was taken into account.

4.4.2.2 Internal Operation & Technologies

Anomalous trajectories traced by drivers often exhibit unusually high or low speed. Using this assumption TAD is able to detect anomalous trajectories such as vehicles performing U-Turns and stopping on the road, as well as vehicles moving at dangerously high speeds. Instead of the Z-score, the TAD component makes use of the 98th and 2nd percentile to find anonymously high speeds. This statistical anomaly detection method is more robust to large outliers in the data than the previously used z-score. The normal ranges for speeds were also determined per path taken since it is common that at junctions the side roads will have lower average speed than the main road. Finally, the TAD component was fitted with a detector error metric, which reduces the number of false detections due to ID switches from the detector. This metric is based on the average distance between each point taken by the detector, normalised to account for perspective. The live update strategy for the TAD was not used for the R2 integration as the normal speed range was found to not change significantly whilst the update strategy slowed down the component. More information about the TAD component and its live update strategy in R1 is included in Section 4.4.2.2 of D5.4.

4.4.2.3 Role in R2 and associated use cases

The TAD component will be used in three of the four use cases implemented in the GRN pilot.

- GRN2: The TAD component will use the CATFlow output to detect trajectories with anomalous speeds.
- GRN3 use case will make use of TAD to monitor the traffic state and infer anomalous events in real time (contributing to reaching the goal of detecting anomalous events within 2 minutes of the events taking place).
- GRN4 use case will make use of the TAD for its anomalous vehicle speed detection component and collect long term data on anomalous speeds from vehicles.

4.4.3 Visual Anomaly Detection (ViAD)

4.4.3.1 Overview

The visual anomaly detection component detects situations that are either considered novel for a visual scene based on a sequence of frames or belong to a set of anomalies predefined in the training set. As an input, it takes a sequence of video frames and produces a label for that sequence, declaring it normal or anomalous. Additionally, it could be configured to provide an anomaly heatmap.

The ViAD component is present in the MT1 use case. The model is trained for a specific camera, and it can recognise different kinds of events as anomalies, based on the data and requirements provided by each use case owner. In the weakly-supervised training, the data is provided in the form of various length clips that are classified in their entirety as a certain event. This approach, while more time-consuming for the data providers, generally leads to improved performance. Depending on the required operational speed, the component can run on a system with a dedicated GPU, which provides for faster operation, or on a CPU.

4.4.3.2 Internal Operation & Technologies

The ViAD component is a deep neural network trained in a weakly-supervised learning manner. The data pre-processing within the component is the same as in YOLO-SED (Section 4.4.11). The model is implemented in Python using PyTorch and PyTorch Lightning frameworks and is based on the Anomaly Regression Net (AR-Net)[33]. The model architecture is presented in Figure 31.

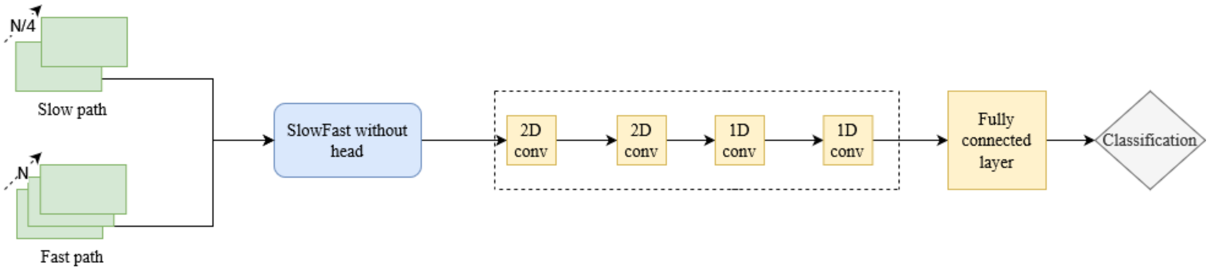


Figure 31: ViAD model architecture

During the training procedure, the model is presented with normal and anomalous situations. It learns to distinguish between them by employing two loss functions: Dynamic Multiple-

[33] Wan, B., Fang, Y., Xia, X., & Mei, J. (2020). Weakly supervised video anomaly detection via center-guided discriminative learning. Proceedings - IEEE International Conference on Multimedia and Expo, 2020-July

Instance Learning Loss (DMIL) and Center Loss (CL). The DMIL aims to enlarge the inter-class distance between normal and anomalous situations, while CL minimises the intra-class representations of normal instances. In other words, the DMIL helps the model to differentiate between the different types of anomalies and normal instances, while CL makes sure that the representations of normal situations are kept close to each other. In this way, the ViAD model is able to learn a solid understanding of what constitutes normal situations and, at the same time, tries to maximise the difference between them and any potential anomalies it encounters. The SlowFast visual backbone is not trained, i.e., a pre-trained, publicly available model is used as a feature extractor from the input sequence.

4.4.3.3 Role in R2 and associated use cases

In R2, a weekly-supervised visual anomaly detection method was developed. This method was integrated to the ViAD component in the MARVEL framework.

ViAD is used in the MT1 use case.

4.4.4 Audio-Visual Anomaly Detection (AVAD)

4.4.4.1 Overview

The audio-visual anomaly detection component detects situations that are considered novel for a visual scene based on a sequence of frames and audio. As an input, it takes a window of video frames together with the corresponding audio snippet and produces the scores indicating how likely it is for each frame to include a novelty. Additionally, it could be configured to provide an anomaly heatmap.

The AVAD component is present in multiple MARVEL use cases. In all of them, it works in the same way, the only difference being that for each use case, a tailor-made model is created. These models are each trained for a specific camera, and they can recognise different kinds of events as anomalies, based on the data and requirements provided by each use case owner.

The component has been built and implemented with a weakly-supervised model. This means that the visual part of the model is largely similar to ViAD, described in Section 4.4.3.1. The pre-processing within the component is the same as YOLO-SED (Section 4.4.11). The main enhancement upon the ViAD model stems from the AVAD's ability to ingest both visual and auditory data. The data are provided to the model in the form of a sequence of frames, along with corresponding audio snippets. Those audio snippets have a length of 1 second, and they contain the auditory information preceding, and including, their corresponding frame timing.

4.4.4.2 Internal Operation & Technologies

The model is implemented in Python using PyTorch and PyTorch Lightning frameworks. Its implementation is based on the works by Wan et al.[34] and Chumachenko et al.[35].The model architecture is presented in Figure 32.

[34] Wan, B., Fang, Y., Xia, X., & Mei, J. (2020). Weakly supervised video anomaly detection via center-guided discriminative learning. *Proceedings - IEEE International Conference on Multimedia and Expo*, *2020-July*

[35] Chumachenko, K., Iosifidis, A., & Gabbouj, M. (2022). Self-attention fusion for audiovisual emotion recognition with incomplete data. *Proceedings - International Conference on Pattern Recognition*, *2022-Augus*, 2822–2828.

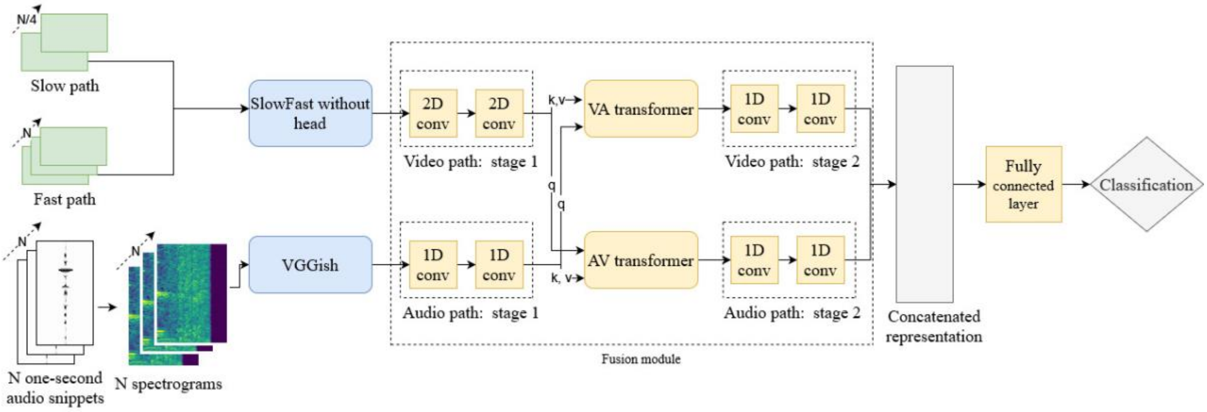


Figure 32: AVAD model architecture

AVAD introduces a parallel audio track to the ViAD architecture, as well as a couple of transformers that fuse the information from the audio and visual modalities. The transformers help the model to decide which parts of the visual input should pay attention to which parts of the audio input, and vice versa. Then those paths are concatenated, and a final label is produced for the incoming data. The training procedure is the same as for the ViAD model (Section 4.4.3.2). The visual backbone is not trained, as in ViAD (Section 4.4.3.3). Similarly, for the VGGish backbone a pre-trained public model is used.

4.4.4.3 Role in R2 and associated use cases

In R2, a new methodology was developed and tested on the new audio-visual anomaly detection dataset created by MARVEL. This method was integrated to the AVAD component in the MARVEL framework.

The AVAD component is applied to use cases GRN3 and MT3.

4.4.5 Visual Crowd Counting (VCC)

4.4.5.1 Overview

The visual crowd counting component estimates the number of people present in a scene, based on a single image (or video frame) as an input. Additionally, it is capable of producing a heatmap depicting the localisation of the detected people.

The VCC component is present in multiple MARVEL use cases. In all of them, it works in the same way, the only difference being that for each use case, a tailor-made model is created. These models are each trained for a specific camera, to optimise their performance for the specific camera positions and recorded scenes.

The VCC component is trained in a supervised manner. It requires fully labelled data, i.e., frames that represent as varied operational environment as possible, each annotated with points marking the centre of head of each person present in the frame.

Depending on the required operational speed, the component can run on a system with a dedicated GPU, when it is faster, or a system without one. In the case of VCC, the impact of the lack of GPU is quite significant.

4.4.5.2 Internal Operation & Technologies

The VCC component is based on deep learning. It is a deep neural network which is trained in a supervised learning manner. The current implementation is using neural network (backbone) architecture called VGG16 network. The network is implemented in Python in Keras and Tensorflow or in Pytorch.

Currently, VCC is implemented using the AVCC architecture (Section 4.4.6), with a tensor filled with zeros acting as the audio input.

The component spawns 3 processes during its operation as of M16: one process takes care of receiving and queuing incoming RTSP data, one process prepares the received visual data to match required format, and the last spawned process runs the model inference.

The pre-processing within the component is the same as YOLO-SED (Section 4.4.11). The inference results for each frame are sent in an appropriately formatted manner as a JSON file to the corresponding MQTT broker.

4.4.5.3 Role in R2 and associated use cases

The component is applied in the MT1 use case, where it is deployed in the Cloud. VCC is also applied in the UNS1 use case, where it is deployed at the Fog. VCC is connected with other components as follows:

- VCC receives data from the Data Corpus;
- VCC can be trained using DynHP;
- VCC sends its output to DatAna and SmartViz.

VCC's contribution to the applicable use cases is the estimation of number of people per video frame in real or near real-time.

4.4.6 Audio-Visual Crowd Counting (AVCC)

4.4.6.1 Overview

The audio-visual crowd counting component estimates the number of people present in a scene, based on a single image (or video frame) and a 1-second audio snippet as an input. Additionally, it is capable of producing a heatmap depicting the localisation of the detected people.

The AVCC component is present in multiple MARVEL use cases. In all of them, it works in the same way, the only difference being that for each use case, a tailor-made model is created. These models are each trained for a specific camera, to optimise their performance for the specific camera positions and recorded scenes.

The AVCC component is trained in a supervised manner. It requires fully labelled data, i.e., frames that represent as varied operational environment as possible, each annotated with points marking the centre of head of each person present in the frame.

Depending on the required operational speed, the component can run on a system with a dedicated GPU, when it is faster, or a system without one. In the case of AVCC, the impact of the lack of GPU is quite significant.

4.4.6.2 Internal Operation & Technologies

The AVCC component is based on deep learning. It is a deep neural network which is trained in a supervised learning manner. The AVCC component's visual backbone is based on VGG16 network[36]. The audio backbone uses VGG-ish network for feature extraction. Additionally, a

[36] https://neurohive.io/en/popular-networks/vgg16/

fusion block, using 2D convolutions, is applied to merge the outputs of the video and audio paths before making the final prediction. The network is implemented in Keras and Tensorflow.

The currently implemented AVCC model is based on the AudioCSRNet [37](Hu, 2020). The method utilises the first 10 layers of the pre-trained VGG for the visual path and VGGish network as the backbones of the visual and audio paths. The features extracted by those networks are then fused using six dilated convolution layers. By using pre-trained backbones, the network is able to be finetuned using audio-visual datasets that tend to be much smaller than purely visual or audio datasets. The architecture of AudioCSRNet is presented in Figure 33.

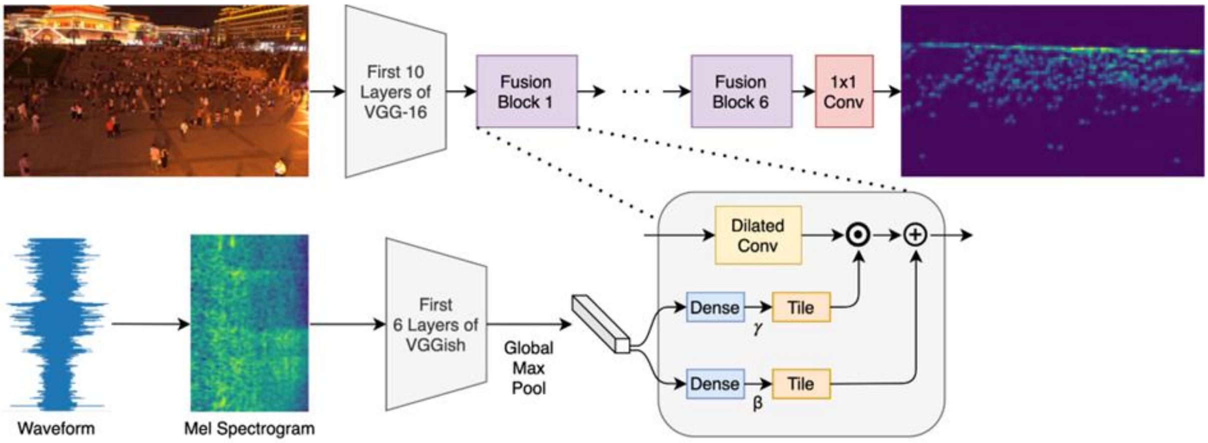


Figure 33: Architecture of the AudioCSRNet

The component spawns 4 processes during its operation as of M16: one process takes care of receiving and queuing incoming RTSP data, two processes prepare the received audio-visual data to match required format, and the last spawned process runs the model inference.

The pre-processing within the component is the same as YOLO-SED (Section 4.4.11). The inference results for each frame are sent in an appropriately formatted manner as a JSON file to the corresponding MQTT broker.

4.4.6.3 Role in R2 and associated use cases

The component is unchanged since the MVP. Once labelled data are available, it can be trained or finetuned for specific use cases.

The component is applied in the GRN4 use case, where it is deployed at the Cloud. AVCC is connected with other components as follows:

- AVCC receives data from the Data Corpus;
- AVCC can be trained using DynHP;
- AVCC can be trained using FedL;
- AVCC sends its output to DatAna and SmartViz.

AVCC’s contribution to the GRN4 use case is the estimation of number of people per video frame in real or near real-time.

[37] Hu, D., Mou, L., Wang, Q., Gao, J., Hua, Y., Dou, D., & Zhu, X. (2020). Ambient Sound Helps: Audiovisual Crowd Counting in Extreme Conditions. ArXiv, abs/2005.07097.

4.4.7 Sound Event Detection (SED)

4.4.7.1 Overview

The sound event detection (SED) component enables the detection of sound events and their temporal location in the audio signal. Acoustic environments in smart cities are full of sounds which provide important information for understanding what is happening in the environment. Humans have formed tight associations between events in the scene and the sounds these events produce. These associations are called sound events and they are represented as a textual label.

Sound event detection is used in the various use cases of MARVEL to offer the ability to detect actions and events based on captured audio signal. The targeted sound events are selected based on the requirements of the use cases. The detection of these sound events can be used as standalone information in the scene analysis or as complementary information to other systems to gain deeper understanding of the scene. SED provides the ability of detecting activities in the scene, which is a core functionality of the audio perception in MARVEL. An overview of the sound event detection process is shown in Figure 34.

For R2, the component has been extended from R1 by adding transfer learning stage into the training process to increase the generalisation ability of the model.

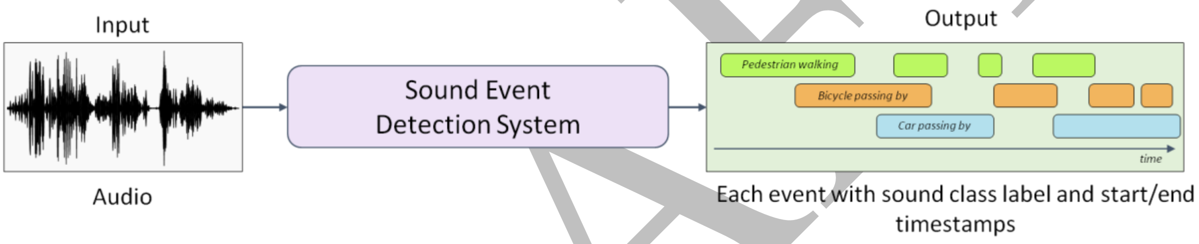

Input
Audio
Sound Event Detection System
Output
Pedestrian walking
Bicycle passing by
Car passing by
time
Each event with sound class label and start/end timestamps

Figure 34: Overview of a sound event detection

The input for this component is an audio signal, and the component provides activity prediction of the sound events in pre-specified units of time in the output. To train the component, a strongly labelled audio dataset recorded in each specific use case environment is required. The dataset should contain examples from all sound events targeted in the use case. The component was developed to run on all levels of the MARVEL infrastructure, in a high-performance CPUs in cloud computing layers as well as in nodes in fog and edge layer with GPU. The component was designed to operate in real time setup continuously consuming real-time audio-visual stream.

4.4.7.2 Internal Operation & Technologies

The SED component is implemented in Python language, and the AI functionality is built on the PyTorch machine learning framework. All audio AI components from TAU (SED, AT, AAC, and SELD) share the same code base, and they contain mostly similar data processing steps. Component-specific processing steps are added to the default data processing pipeline. Overview of the component released in R1 was given in D5.4 (Section 4.4.7.2), and in R2 the component internal operation was updated to allow deployment to GPU-enabled nodes in the MARVEL infrastructure.

The component is designed to work with continuous audio stream in real-time setting, and thus the component has three processes working in parallel in a multi-threaded manner. The receiver process is responsible for receiving and extracting audio segments from the real-time audio-visual stream (RTSP stream), and the AI process is responsible for applying sound event detection for an audio segment. The transmitter process is responsible for preparing the detection output along with meta-information about the source and AI model and transmitting information in a JSON structure to the receiving MQTT broker. The receiver process is using FFMPEG software to consume the RTSP stream and extract audio from the stream. Extracted audio is moved to the receiver process through Linux STDOUT pipe (standard output). An overview of the internal structure of the component is shown in Figure 35.

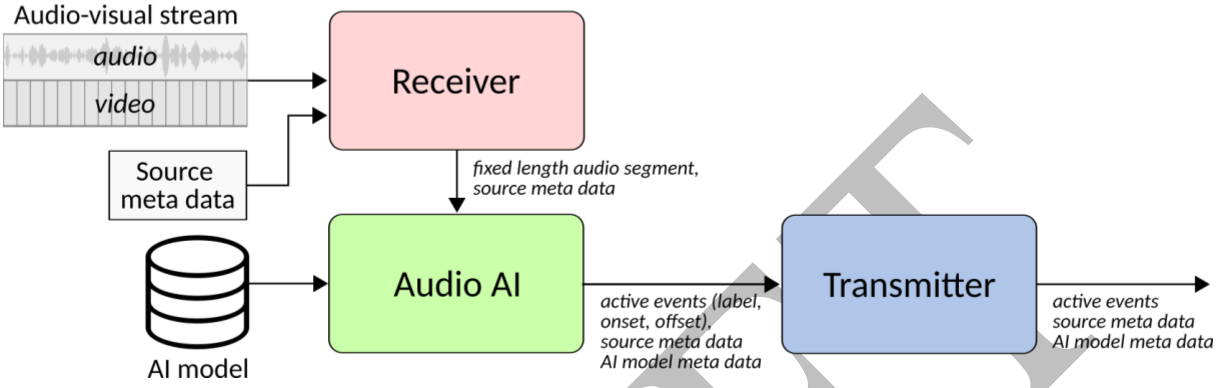

Audio-visual stream
audio
video
Source meta data
AI model
Receiver
fixed length audio segment, source meta data
Audio AI
active events (label, onset, offset), source meta data AI model meta data
Transmitter
active events source meta data AI model meta data

Figure 35: Overview of processes in the real-time sound event detection component

The audio AI process is loading an AI model trained with use case specific data using a supervised learning approach. The AI model is a neural network based on an architecture with four convolutional layers and two fully connected layers. Detection is applied in 1-second analysis segments. Further details on audio AI subsystem can be found in D3.1[38] and D3.5.

4.4.7.3 Role in R2 and associated use cases

The SED component is applied in the R2 release in GRN2, GRN4, MT2, MT3, and MT4 use cases. The AI model for the component was trained with use case specific data from Data Corpus, and the trained models were stored in the AI Model Repository. In the inference phase, the pre-trained AI model is loaded into the component from the AI Model Repository. The sound event detection output from the component is sent to DatAna.

4.4.8 Audio Tagging (AT)

4.4.8.1 Overview

The audio tagging (AT) component enables the recognition of the sound source activity inside audio segments with predefined fixed lengths. This functionality is used in the use cases of MARVEL to offer the ability to recognise sounds related to actions or events with coarse time resolution. The sound classes to be recognised are dependent on each use case's specifications. The information about the sound class activity can be used as standalone information or as complementary information to other systems to gain deeper understanding of the scene. An overview of the audio tagging system is shown in Figure 36.

For R2, the component has been extended from R1 by adding transfer learning stage into the training process to increase the generalisation ability of the model.

[38] "D3.1: Multimodal and privacy-aware audio-visual intelligence – initial version," Project MARVEL, 2022. https://doi.org/10.5281/zenodo.6821318.

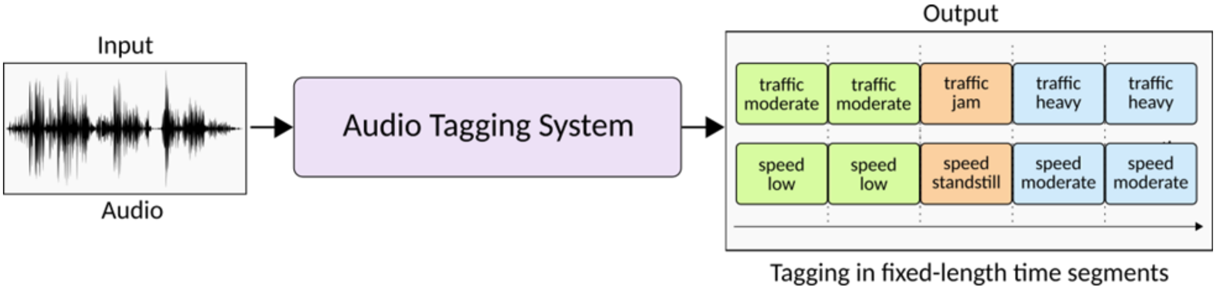


Figure 36: Illustration of an audio tagging system that can recognise simultaneously class related to traffic amount and class related to overall traffic speed

4.4.8.2 Internal Operation & Technologies

The audio tagging component shares the code base with all audio AI components from TAU. The structure of the component was explained in Section 4.4.7. The component uses an AI model trained with use case specific data from Data Corpus using a supervised learning approach. The AI model is a neural network based on an architecture with four convolutional layers and two fully connected layers. The data processing pipeline is very similar to the SED component; however, audio tagging is producing sound activity aligned with the 5-second analysis segment boundaries. Overview of the component released in R1 was given in D5.4 (Section 4.4.7.2), and in R2 the component internal operation was updated to allow deployment to GPU-enabled nodes in the MARVEL infrastructure. Further details on audio AI subsystem can be found in D3.1 and D3.5.

4.4.8.3 Role in R2 and associated use cases

The AT component had been applied in GRN2 in the R1 release and in R2, it is additionally applied in the MT2 and MT3 use cases. The AI model for the component was trained with usecase specific data from Data Corpus, and the trained models were stored in the AI Model Repository. In the inference phase, the pre-trained AI model is loaded into the component from the AI Model Repository. The sound event detection output from the component is sent to DatAna.

4.4.9 Automated audio captioning (AAC)

4.4.9.1 Overview

The automated audio captioning (AAC) component creates textual description with full sentences for each audio segment. The caption will describe what is happening in the audio signal, for example, “people yelling while siren wails”. These captions can be used as a direct description for humans accessing audio-visual streams, as well as for further text-based analysis to assist the decision-making process. This component is used in the monitoring use cases in the MARVEL project to provide descriptive captions of audio-visual segments accessed by the monitoring system users. An overview of the audio tagging system is shown in Figure 37.

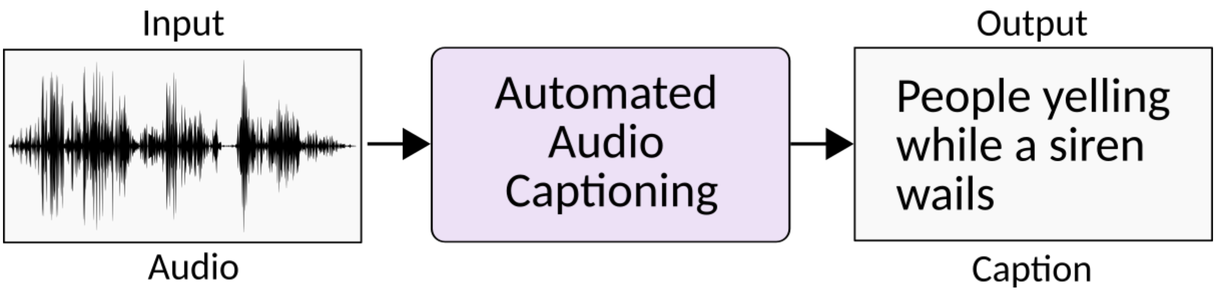

Input
Audio
Automated Audio Captioning
Output
People yelling while a siren wails
Caption

Figure 37 Overview of automates audio captioning system

4.4.9.2 Internal Operation & Technologies

The automated audio captioning component shares the code base with all audio AI components from TAU. The structure of the component was explained in Section 4.4.7. The component uses an AI model trained with use case specific data from Data Corpus using a supervised learning approach. The AI model is a neural network based on an architecture with convolutional encoder and a transformer decoder. The data processing pipeline is very similar to AT component; however, AAC is producing captions that are aligned with the 5-second analysis segment boundaries. Further details on audio AI subsystem can be found in D3.1 and D3.5.

4.4.9.3 Role in R2 and associated use cases

The AAC component is applied in the R2 release in MT2 use case. The AI model for the component was trained with use-case specific data from Data Corpus, and the trained models were stored in the AI Model Repository. In the inference phase, the pre-trained AI model is loaded into the component from the AI Model Repository. The captioning output from the component is sent to DatAna and used by the GPURegex component for further text-based analysis.

4.4.10 Sound event localisation and detection (SELD)

4.4.10.1 Overview

Sound event localisation and detection (SELD) task is a joint task where a system jointly performs sound event localisation and sound event detection. The localisation detects the directional characteristics of the sound events by outputting the azimuth and elevation of the direction of arrival of the sound that is classified as belonging to a sound event. The localisation is performed with respect to the microphone position and orientation. The SELD system can be used to produce an in-depth view of the auditory scene in MARVEL use cases where sound localisation is an important aspect. An overview of the system is shown in Figure 38.

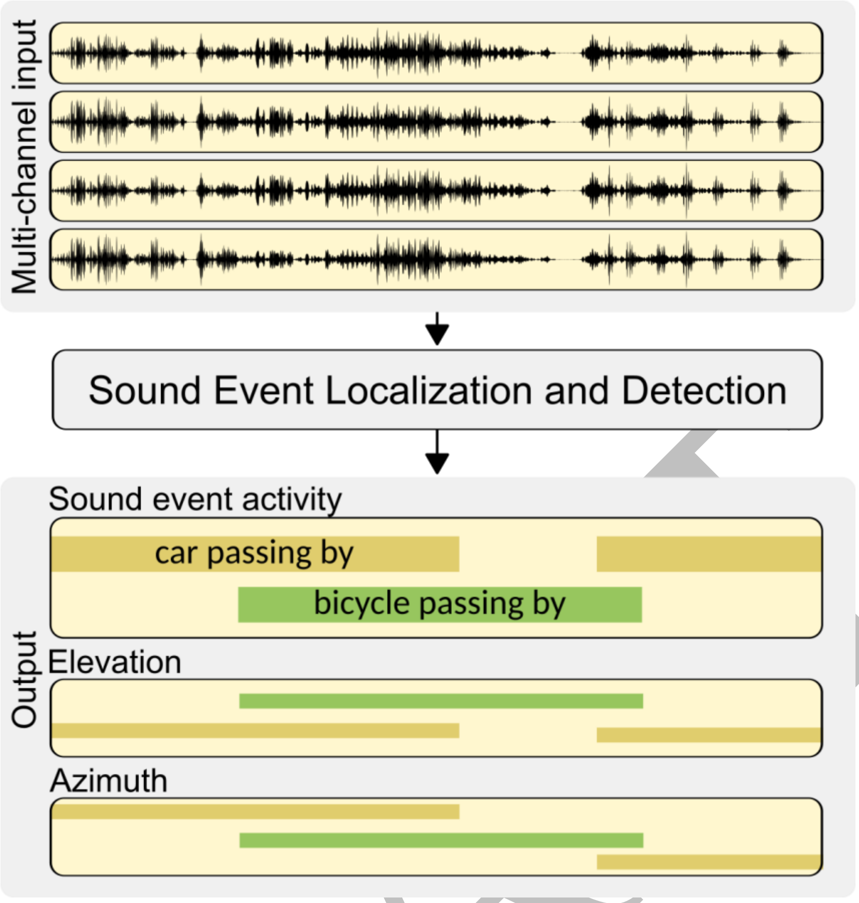


Figure 38: Sound event localisation and detection system

4.4.10.2 Internal Operation & Technologies

The sound event localisation and detection component share the code base with all audio AI components from TAU. The structure of the component was explained in Section 4.4.7. The component uses an AI model trained with use case specific data from Data Corpus using a supervised learning approach. The AI model is a neural network based on an architecture with convolutional layers, recurrent layers, and multi-head self-attention blocks. The data processing pipeline is very similar to the SED component, but the SELD component is producing per event also azimuth and elevation estimates. Further details on audio AI subsystem can be found in D3.5.

4.4.10.3 Role in R2 and associated use cases

The SELD component is applied in the R2 release in UNS2 use case. The AI model for the component was trained with use case specific data from Data Corpus, and the trained models were stored in the AI Model Repository. In the inference phase, the pre-trained AI model is loaded into the component from the AI Model Repository. The sound event localisation and detection output from the component is sent to DatAna.

4.4.11 YOLO-SED

4.4.11.1 Overview

The YOLO-SED component is a part of the GRN1 use case, aimed at analysing audio-visual data on the edge and detecting anomalies using YOLO object detector and SED audio analysis module. The outputs of these modules are fused together, and the anomaly prediction is sent to an MQTT broker. The SED subsystem detects sound event activity from the given audio segment.

4.4.11.2 Internal Operation & Technologies

The operation of the component should be performed on an NVIDIA Jetson NX platform with 6 CPU cores, 8 GB shared VRAM/RAM and JetPack 4.6.2. The Python interpreter available on this device is Python 3.6. The YOLO-SED component can be deployed directly on the device, or in a containerised form using Docker.

The structure of YOLO-SED is presented in Figure 39. YOLO-SED receives two RTSP streams for video and audio. Both streams are consumed by an FFmpeg sub-process and piped inside Python with process-level communication. For each stream, there are two pipes that are used: one for the data and another one for the timestamps. This ensures that the timestamps are clock-independent and synchronised with data. In case of a failure of any of the RTSP streams, the corresponding message is sent to a controller thread which stops and restarts all the streams. New streams have an additional timestamp offset which ensures that old and new data cannot be processed together. After video and audio data samples are created by the corresponding readers, they are consumed by a pre-processing thread which pairs each image with a corresponding 1s audio clip and applies pre-processing for these samples. Video pre-processing includes image cropping and resizing, while audio pre-processing creates a PyTorch tensor of a Numpy audio sample.

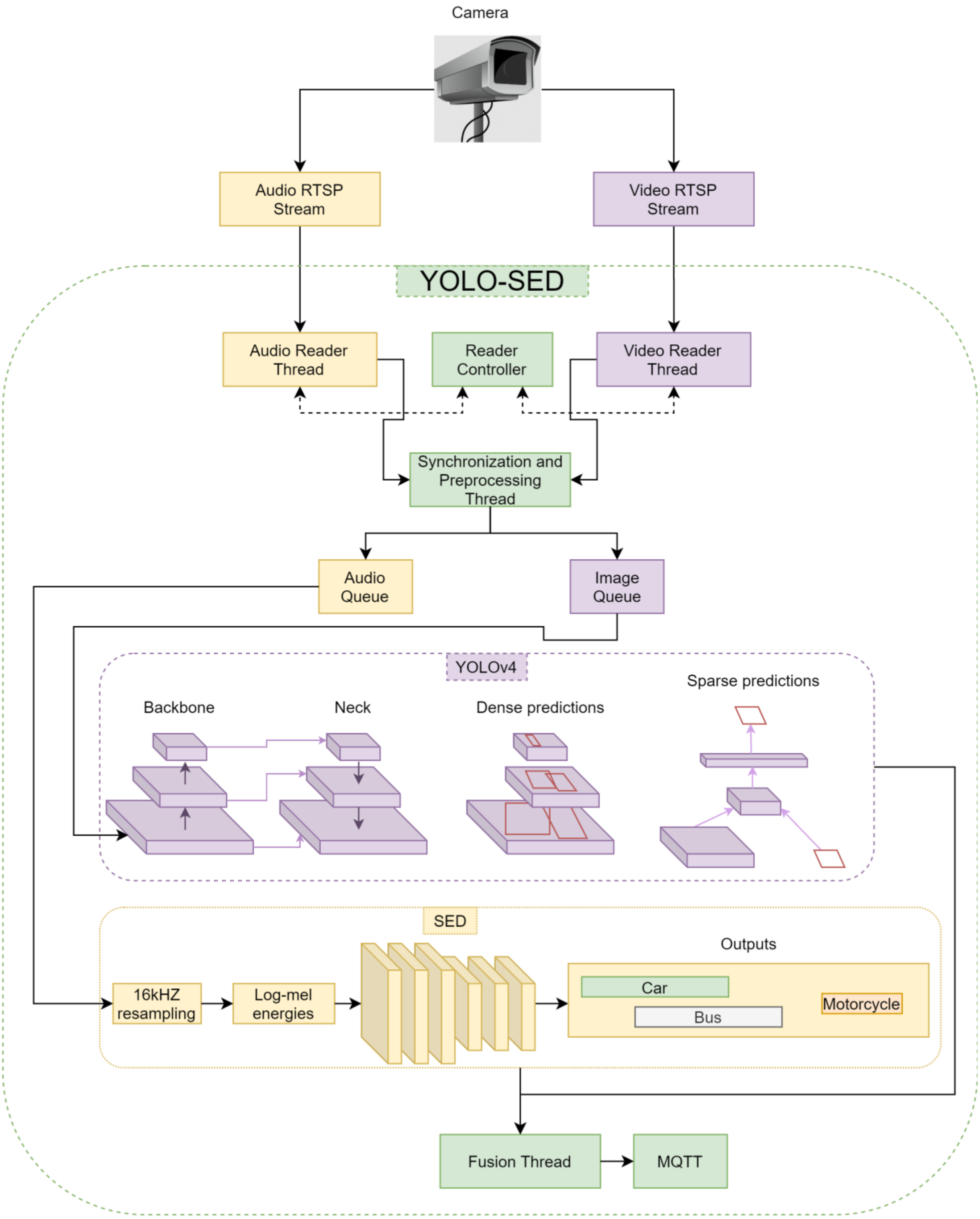


Figure 39: Structure of YOLOSED

After pre-processing, each audio-visual pair is identified by the same unique id and split to be processed by both YOLO and SED in corresponding machine learning threads. The YOLO thread consumes images and outputs object detections, filtered out based on the vulnerability status of each class. The correct class may not be detected perfectly for similar classes. For this reason, the SED component analyses audio samples and outputs predicted classes from audio.

The predictions are combined together to provide more certain class and vulnerability predictions. The anomaly is detected if at least one vulnerable object is detected.

In the SED subsystem, the processing pipeline and the model architecture are similar to the SED component (see Section 4.4.7). The SED subsystem uses an AI model trained with use case specific data from the Data Corpus using a supervised learning approach. The trained model is stored in the AI Model Repository and loaded when the YOLO-SED component is started.

4.4.11.3 Role in R2 and associated use cases

The YOLO-SED component is a part of the GRN1 use case. It is deployed on the GRNEDGE3 device and sends data to a DatAna edge node which communicates with an LED sign via Arduino.

4.4.12 Rule-Based Anomaly Detection (RBAD)

4.4.12.1 Overview

The RBAD is a lightweight, logic-based anomaly detector implemented in Python. It employs a predefined ruleset to detect very specific anomalies, based on the input messages coming from the CATFlow component, as described in Section 4.4.1. The input messages contain information about the objects detected in the video frame, as well as their location and time of detection. Based on this information, combined with the predefined rules, RBAD is able to create a specific anomaly alert. RBAD has been set up to detect the following situations: pedestrians jaywalking, buses arriving not on schedule, heavy-weight vehicles presence during rush hours, and bikers using the pavement.

4.4.12.2 Internal Operation & Technologies

For all of the anomalous situations, described in the previous section, a human expert creates the required rules. For the pedestrians and bikers, a map, of which example can be seen in Figure 40, is produced. It defines the locations in the scene where bikers or/and pedestrians are allowed to be. If CATFlow detects the corresponding class in an undesired location, an anomaly alert is created.

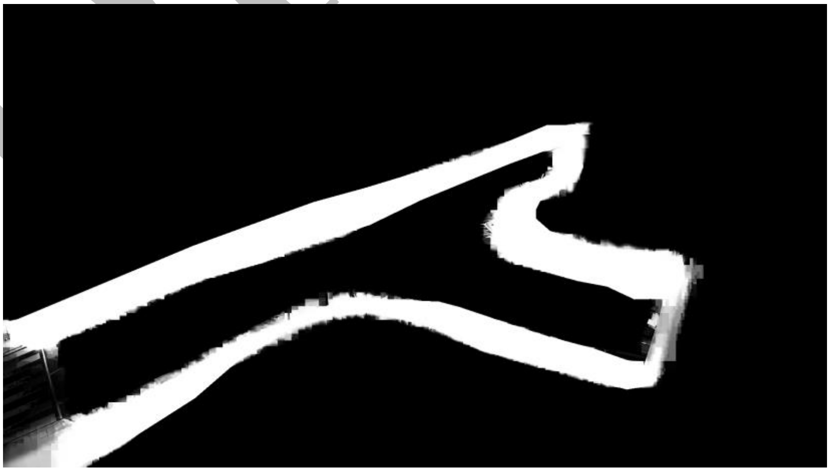


Figure 40: Example pedestrian allowed presence map.

In case of bus anomaly, RBAD contains the bus schedules from the bus stops present in the scene and compares the bus arrival times with the expectations. If the difference is bigger than some predefined amount of time, the situation is recognised as an anomaly. Similarly, the rush hours are defined by a human expert, and any heavy-weight vehicles detected during those hours raise an anomaly alert. The RBAD architecture is illustrated in Figure 41.

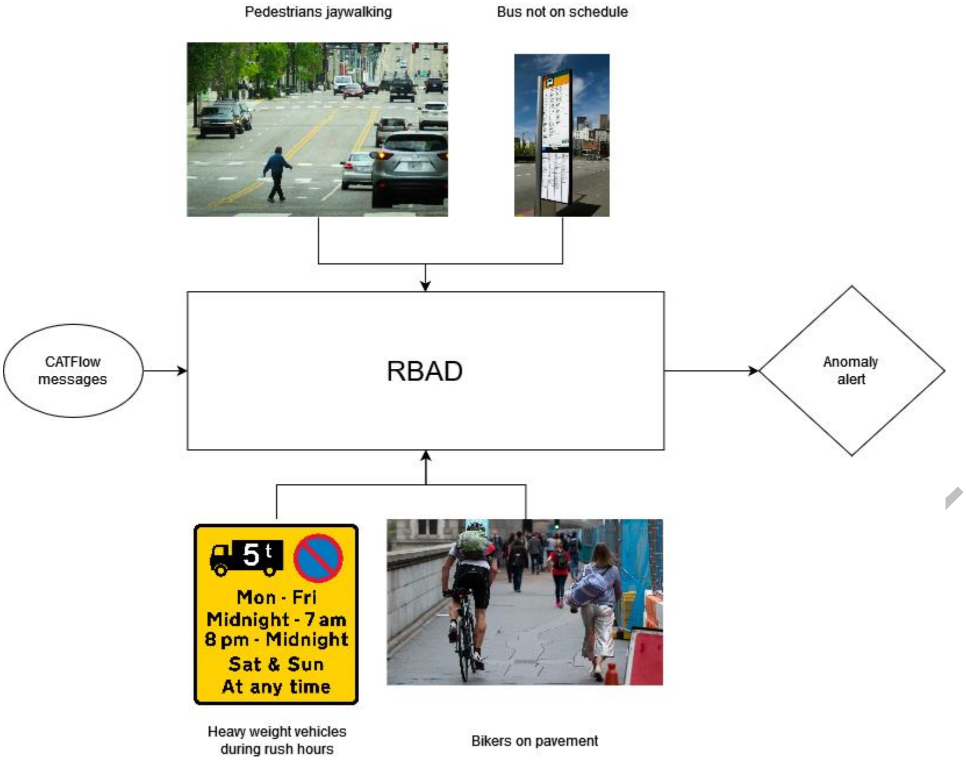


Figure 41: RBAD architecture

RBAD does not require any training, however, it does require input from the end user to help define the rules for the implemented anomaly types.

4.4.12.3 Role in R2 and associated use cases

RBAD was developed and integrated in the context of R2 for the needs of GRN2.

4.5 Optimised E2F2C processing and deployment Subsystem

4.5.1 GPURegex

4.5.1.1 Overview

GPURegex is a GPU-accelerated pattern-matching engine. GPURegex leverages the parallelism properties of general-purpose GPUs (GPGPUs) to accelerate the process of string or regular expression matching. This component was originally designed and implemented in order to accelerate the pattern-matching mechanism that is the core operation of typical networking applications such as network intrusion detection, firewall and filtering Layer 7 from the OSI stack. Now, in the context of the MARVEL project, GPURegex will be used to accelerate the processing of text-based input originating from audio/video (such as captions), against a number of predefined patterns (i.e., keywords). This procedure will be used for near real-time event detection in public environments.

4.5.1.2 Internal Operation & Technologies

GPURegex uses the characteristics of graphics processors to offer parallelism in computing, with ultimate goal the processing acceleration of the workload. To deploy GPURegex, an OpenCL-enabled graphics processor unit is required. The GPU can be either a dedicated, discrete GPU (e.g., a NVIDIA GPU) or an integrated chip within the CPU die (e.g., Intel HD Graphics). Moreover, in the case of GPU absence, the parallelism properties of any OpenCL-enabled CPU could be leveraged. In CPUs, OpenCL takes advantage of the processor's vectorisation instruction set extensions (such as SSE and AVX) offering abundant processing performance, comparable to the performance of a GPU. Apart from the hardware requirements and the need for Docker, OpenCL version 1.2 or newer is required, as well as a new version of CUDA software development kit if the accelerator is provided by NVIDIA.

In addition, GPURegex is based on the Aho-Corasick string searching algorithm[39] and thus, GPURegex performs simultaneous multi-pattern matching against text-based input. The patterns can be either fixed strings or regular expressions.

Different versions of GPURegex can be found in the MARVEL image registry, offering the possibility of deployment in different hardware setups. The first image contains a version of GPURegex that can be executed on an integrated HD Graphics GPU, while the second image enables the execution of GPURegex utilising a CPU (destined for hardware setups that do not contain a discrete GPU). An additional image has been created and uploaded to the MARVEL registry that targets NVIDIA Graphics Processing Units (GPUs).

4.5.1.3 Role in R2 and associated use cases

In R2, GPURegex resides in the fog layer of the MARVEL platform and operates in the context of the MT2 "Detecting criminal/anti-social behaviours" use case. Specifically, GPURegex participates in a pipeline, receiving input from the AAC component and sending its output to SmartViz.

In this pipeline, audio streams originating from the MT microphones are forwarded to AAC for captioning. The resulting captions are published to the DatAna MQTT broker. As GPURegex is subscribed to the corresponding topic, it consumes the output of AAC, right after it is published to DatAna. GPURegex searches for certain keywords as specified by MT, against the AAC captions. GPURegex is intended to detect criminal and anti-social actions that take place in public, in near real-time, using hardware accelerators and parallel processing. When a keyword is matched against the input, GPURegex creates an alert, which is then published to DatAna, with all the required data that describe the event. The output of GPURegex is then visualised by the SmartViz component of MARVEL.

For more information about GPURegex, the reader can also refer to D4.5 *'Security assurance and acceleration in E2F2C framework – final version'*.

4.5.2 DynHP

4.5.2.1 Overview

DynHP is a methodology for pruning deep neural network models at training time. DynHP performs structured pruning in an incremental fashion, i.e., during the training process it explores which groups of parameters are "less informative" and it switches them off

[39] https://dl.acm.org/doi/10.1145/360825.360855

permanently. This kind of "incremental hard pruning" makes this approach suitable for training DNNs on resource constrained devices. Precisely, it is possible to remove the zeroed groups of parameters from the network by re-instantiating it by keeping only the non-zero ones before continuing the training. This allows a real saving in terms of memory during the process. Since the structured hard pruning processes might degrade the performance of the training, DynHP adopts a strategy to contrast it. Precisely, it tunes adaptively the size of the minibatches depending on gradient-related information and the amount of available memory.

Clearly, this process can be used to stop the pruning at any point in time while keeping the training active in order to meet the required performance (e.g. accuracy).

4.5.2.2 Internal Operation & Technologies

DynHP related software library is developed in Python 3 and it significantly relies on the PyTorch Framework.

The creation of a compressed model through DynHP very much resembles the normal training procedure of a DNN.

- Model definition: the model to be compressed is defined using the DynHP primitives, i.e., the layers equipped with the extra parameters used to control the pruning process. In a nutshell, in DynHP we associate to each neuron/filter a gate, i.e., a trainable parameter that ranges. Such a parameter controls the activation probability of the corresponding gate, and the associated probability distribution is the Hard-Concrete distribution[40] (a continuous and differentiable approximation of the Bernoulli distribution). When the activation probability of a gate goes to zero, the gate is considered permanently off, and the associated group of weights can be removed from the DNN.
- Standard Training: the instrumented model is trained and pruned. The output of this process is a pruned DNN model where all the unnecessary parameters are set to zero.

The overall workflow

With respect to R1, it has been redesigned and completely ported for being compatible with Pytorch-lightning, which is a widely used framework for developing, testing and training Deep Neural Network models. In this version, it is possible to set the minimum number of nonzero parameters that should remain active in the network and monitor in real time this quantity. Once a layer reaches the minimum size set during the configuration, the pruning stops for it, while continues for the rest of the network. However, the weights are continuously updated for fine-tuning. In principle, the very same approach can be applied to an already trained model, provided the availability of its definition. Specifically, it is possible to import the parameters of a trained model and use the gates to identify which of them can be pruned. This approach is combined with low bit representation for consistently reducing the footprint of the model.

Finally, the model can be exported without all the additional parameters needed by DynHP for being further deployed using the standard procedures and formats typically used for the deployment of models, (e.g., Torchscript)

[40] Lorenzo Valerio, Franco Maria Nardini, Andrea Passarella, Raffaele Perego, "Dynamic hard pruning of Neural Networks at the edge of the internet," Journal of Network and Computer Applications, Volume 200, 2022, 103330, https://doi.org/10.1016/j.jnca.2021.103330.

4.5.2.3 Role in R2 and associated use cases

In R1, the main focus was on consolidating the methodology, applying it to publicly available benchmarks and, defining a way for interacting with the rest of the MARVEL platform, i.e., DynHP is not meant for being an online service, it needs human interaction for the model definition and instrumentation. The interaction occurs as follows:

- DynHP interacts with the AI Model Repository, retrieving the original model definition.
- Through DynHP, a suitable compression is found. the final compressed model is released. More specifically, the model definition, its parameters and the necessary meta-information are uploaded to the AI Model Repository.
- Finally, all the AI components that want to use the compressed model can download it and use it.

This procedure is persistent regardless of the specific use case, since the only aspects that changes are the data used to train the model and the specification of the model itself.

In R2, DynHP has been applied to Crowd-counting task on the AVCC and VCC models. Since it is an offline tool and it is not meant as a service, it is not directly included in the use cases supported by R2. The performance evaluation performed on benchmark datasets demonstrated that a combined compression based on pruning and mixed precision training produced a model 60% smaller while limiting the performance degradation to 3% over the original model. Such compression allows a significant speedup at inference time (i.e., 4.5x)

4.5.3 FedL

4.5.3.1 Overview

FedL is a component developed by UNS for the MARVEL project's architecture. FedL contains an implementation of a high-performance Federated Learning training model for Deep Learning models. Federated Learning allows for distributed privacy-preserving training of Machine Learning models. In Federated learning, the data never leaves its source, and the training is performed at the data source. Typically, there are multiple Federated Learning clients which perform the training with the local data and a central orchestration point – Federated Learning server which is used to average all the models and provide a global model which is created by combining all the client models. In that way, we obtain a model similar to a model which is trained on all the available client data, but the client data never leave their source. Only the parameters of the Machine Learning models are shared with the server and not the training/input data.

For the needs of the MARVEL project, the FedL component also develops a custom Federated Learning strategy which optimises the learning process for flaky client-server communication. This issue with communication is expected in large heterogeneous systems and the custom model merging strategy is to address it. The custom strategy (names NUS – non-uniform sampling strategy) allows for clients to be temporarily unavailable during learning. It also only requests client training results if the client data are valuable to the global model, based on several metrics such as: number of client data points, model metrics such as accuracy, model gradient variance, client availability history, and others.

4.5.3.2 Internal Operation & Technologies

FedL component is written in Python. It uses the flwr[41] Federated Learning framework as a stable baseline. It provides an API for client-server communication which can be extended to our needs, and this also includes the definition of custom strategies. The baseline strategy Federated Averaging (FedAvg) is used as a baseline for model performance/data throughput testing.

In Federated Learning with FedL gRPC socket communication is used for client-server communication. This way of communication is useful for its stateful nature and tracking of client availability through time (one of metrics used for client sampling).

FedL component is compatible with all major Deep Learning frameworks (e.g., TensorFlow, PyTorch, MXNet) and all the models which can be defined with these frameworks. To enable federated learning only the training loop code needs to be modified, and not the model architecture.

4.5.3.3 Role in R2 and associated use cases

FedL is implemented in the previously defined UNS2 use case Audio-Visual emotion recognition, planned for realisation within the MARVEL R2, with the results reported in D3.5.

4.5.4 MARVDash

4.5.4.1 Overview

MARVdash, a Kubernetes dashboard, simplifies the interaction of domain experts with resources within the E2F2C MARVEL platform by providing a user-friendly and intuitive interface. Instead of directly executing processing tasks, MARVdash focuses on offering efficient deployment mechanisms. It employs a service templating mechanism that allows users to configure and initiate services. Each service is defined by a set of variables, and users can specify values for these variables as execution parameters through the dashboard before deployment. Moreover, MARVdash takes care of configuring various "internal" platform settings, such as the location of a private Docker registry, external DNS name, and other relevant configurations.

4.5.4.2 Internal Operation & Technologies

MARVdash is implemented in Python using the Django framework and is developed and tested on Kubernetes versions ranging from 1.19.x to 1.21.x. It is installed on a virtual machine (VM) deployed on PSNC's OpenStack infrastructure. The VM is equipped with 8 vCPUs, 16 GB RAM, and 512 GB of block storage. It runs on Ubuntu 20.04.1 LTS Linux and utilises Kubernetes version 1.19.8. This VM is situated on the "Cloud" side of the E2F2C platform. For demonstration purposes, the VM is upgraded to 32 vCPUs, 64 GB RAM, and 2 TB of block storage.

MARVdash requires a functional Kubernetes environment with specific features. One essential feature is the certificate management controller, cert-manager, responsible for generating certificates for the admission webhooks. Another necessary feature is an ingress controller that responds to a domain name and its wildcard. Additionally, for storing MARVdash's state, either an existing persistent volume claim or a directory in a shared filesystem mounted at the same

[41] https://flower.dev/

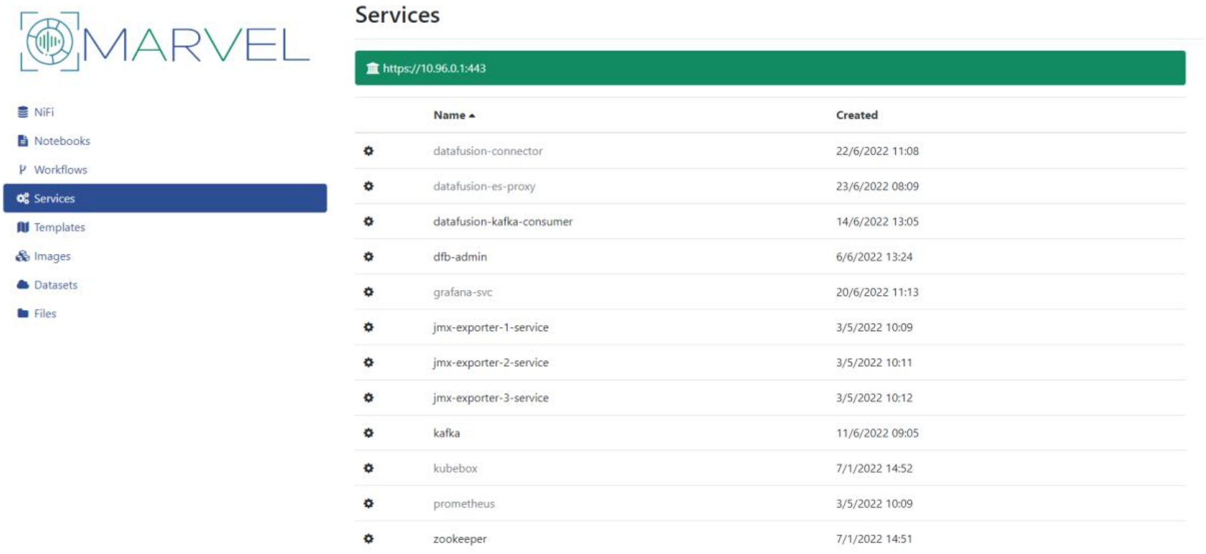
path must be accessible across all Kubernetes nodes. Lastly, a shared filesystem is needed for handling files, similar to the one used for storing the configuration.

To establish a unified Kubernetes cluster comprising nodes from all three layers (Edge, Fog, Cloud), it is crucial for all clients to be connected through a virtual private network, such as EdgeSec-VPN, as described in Section 4.2.4. This setup enables E2F2C communication through MARVdash.

The most recent additions of MARVdash, include Grafana, Prometheus and Loki. Grafana enriches the capabilities of the Kubernetes dashboard by providing a comprehensive set of features. As a robust data visualisation and analytics platform, Grafana empowers users to monitor and analyse the performance and health of their Kubernetes cluster effectively. It offers a diverse collection of customisable dashboards, enabling users to create visually appealing representations of essential metrics such as CPU usage, memory utilisation, and network traffic.

Prometheus is a metrics monitoring tool and Loki serves as a powerful log aggregation system that brings notable advantages to MARVdash. It facilitates 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 originating from different Kubernetes pods, nodes, or namespaces.

MARVdash allows its users to manage the deployment of their components through a UI built for this purpose (Figure 42).



Services

https://10.96.0.1:443

	Name ▲	Created
⚙	datafusion-connector	22/6/2022 11:08
⚙	datafusion-es-proxy	23/6/2022 08:09
⚙	datafusion-kafka-consumer	14/6/2022 13:05
⚙	dfb-admin	6/6/2022 13:24
⚙	grafana-svc	20/6/2022 11:13
⚙	jmx-exporter-1-service	3/5/2022 10:09
⚙	jmx-exporter-2-service	3/5/2022 10:11
⚙	jmx-exporter-3-service	3/5/2022 10:12
⚙	kafka	11/6/2022 09:05
⚙	kubebox	7/1/2022 14:52
⚙	prometheus	3/5/2022 10:09
⚙	zookeeper	7/1/2022 14:51

Figure 42: MARVdash UI

4.5.4.3 Role in R2 and associated use cases

MARVdash serves as a central orchestrator in R2 and all use cases, facilitating the deployment and execution of data management platforms, AI components, and other software components within the E2F2C testbed. It also manages external access to services requiring exposure outside the MARVEL infrastructure while considering connectivity and authentication requirements.

MARVdash allows the deployment of all the components of the Data Management and Distribution (DMP) subsystem of the MARVEL framework. DatAna (Apache NiFi, Mosquitto MQTT, and NiFiRegistry), Data Fusion Bus (Kafka, Elastic Search, Kibana), StreamHandler, Hierarchical Data Distribution (HDD) are deployed in different layers (edge, fog, cloud) and targeted worker nodes.

Regarding the User interactions and decision-making subsystem, MARVdash eases the deployment of SmartViz and Data Corpus. The integration of web proxy services was necessary for HTTPS-compliant URLs ensuring the encryption of the exchanged data. Moreover, the authentication mechanism of MARVdash was utilised.

As far as the Audio, Visual, and Multimodal AI subsystem is concerned, the whole set of AI MARVEL components is deployed through MARVdash. The simplified deployment mechanism that is offered, allows the component owners to select their installation target based on their needs on CPU/GPU availability. It also offers private and shared files for model storage.

Our goal regarding MARVdash functionality was to ease the deployment of every MARVEL component of every MARVEL subsystem, in any cluster layer, in any MARVEL use case. The only prerequisite is the target deployment environment to be a node that is part of the MARVEL Kubernetes cluster.

For further details please refer to D3.6 *'Efficient deployment of AI-optimised ML/DL models– final version'*.

4.6 E2F2C Infrastructure

4.6.1 HPC infrastructure

4.6.1.1 Overview

The main goal of this component is to ensure the high performance of data processing in the last level of the E2F2C stack. In the MARVEL project, this is provided by PSNC, which offers novel HPC infrastructure (Eagle cluster) and virtualised private cloud (LabITaaS) with an OpenStack web interface tool allowing for a flexible allocation of computing resources and various class storage services connected with the HPC system. Details were already presented in D5.3, which was a report on the HPC infrastructure and resource management aspects for the first integrated version of the MARVEL framework. The final version of HPC Infrastructure will be described at the end of the project in D5.8.

4.6.1.2 Internal Operation & Technologies

From MARVEL's perspective, the essential part of the HPC infrastructure is virtualised private cloud, which serves as a base for deploying software stack and storing project data. Cloud resources come from two geographically distributed regions of PSNC – DCW and BST. To sum up, resources available exclusively for the MARVEL project include nearly 350 VCPUs, more than 500GB of RAM and about 1PB of storage. Apart from that, PSNC offers the Eagle supercomputer with tens of NVIDIA V100 GPUs. Compared to R1, the significant improvement is the development of a direct connection between cloud infrastructure and Eagle supercomputer, which enables extending the MARVEL Kubernetes cluster to the GPU node. To this end, one of Eagle nodes has been dedicated to MARVEL and VM (with direct access to the graphics cards) has been installed on it using the libvirt toolkit. The provided solution enables the deployment of components that require constant access to a GPU and allows for two-sided direct data access between MARVEL components deployed on different infrastructures.

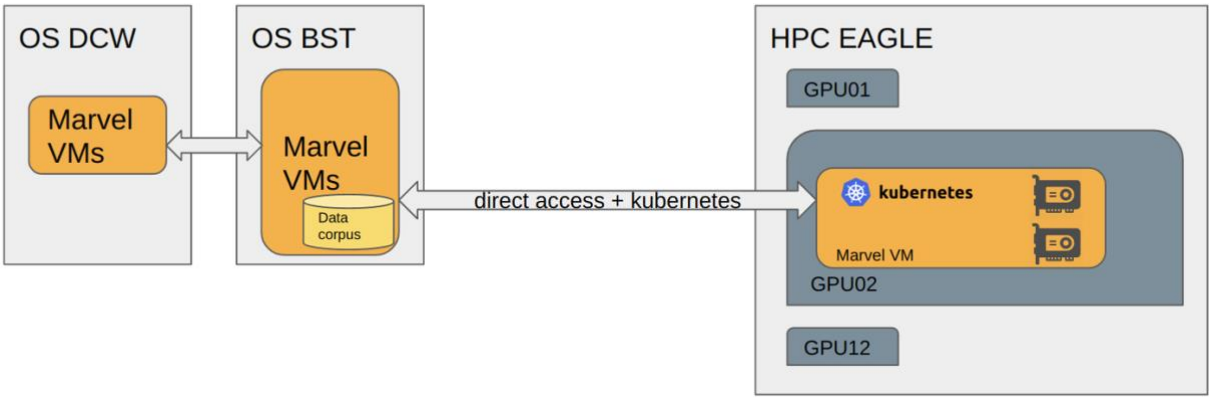


Figure 43: Connection between cloud and HPC infrastructure

4.6.1.3 Role in R2 and associated use cases

Provided infrastructure plays an important role in R2 as it enables efficient deployment of the cloud layer of the MARVEL E2F2C framework. Consequently, it applies to all use cases where architecture requires running some component instances in the cloud layer. Furthermore, delivered infrastructure allows for storing significant amounts of data collected by the MARVEL Data Corpus.

4.6.2 Management and orchestration of HPC resources

4.6.2.1 Overview

The task of this component is to ensure high performance of computing, storage and network resources provided by PSNC. Details were already presented in D5.3, which was a report on the HPC infrastructure and resource management aspects for the first integrated version of the MARVEL framework. The final version of the management and orchestration of HPC resources will be described at the end of the project in D5.8.

4.6.2.2 Internal Operation & Technologies

The aspects of management and orchestration refer to actions taken by PSNC specialists from various IT fields to ensure the efficiency of the delivered HPC resources. PSNC contributes to the MARVEL framework development process by providing technical support from experts in the areas of HPC, cloud, storage and security.

From a technological point of view, an important tool for MARVEL administrators is the OpenStack web interface which is deployed as Infrastructure-as-a-Service and allows for a flexible allocation of computing resources and various class storage services connected with the HPC system. Apart from that, all MARVEL project participants can use Eagle with the SLURM queuing system to perform tasks that require high computing power, e.g., to train AI models with the support of GPUs.

After R1, PSNC collaborated with FORTH to develop the Zabbix monitoring system dedicated to the MARVEL project. The delivered solution enables the collection of detailed statistics related to individual components of the MARVEL project. PSNC also disposes of a set of tools to monitor the general availability of Cloud and Eagle infrastructure, which has already been described in Section 8.6. of D6.2[42].

[42] "D6.2: Evaluation Report," Project MARVEL, 2022. https://doi.org/10.5281/zenodo.7296312.

4.6.2.3 Role in R2 and associated use cases

As a technology partner, PSNC provides technical support, ensures the proper functioning of offered services, and resolves any technical issues arising during the MARVEL framework deployment on PSNC infrastructure.

4.7 User interactions and decision-making toolkit

4.7.1 SmartViz

4.7.1.1 Overview

SmartViz is a data visualisation toolkit that constitutes the User Interface (UI) of the Decision-Making Toolkit (DMT). It provides the means to visualise numerous incoming detected events and anomalies deriving from the analysis and the evaluation of data of all the MARVEL use cases. SmartViz consists of a set of visualisation tools developed to allow exploratory analysis of data by using interactive representations, features, and visualisation widgets.
In this deliverable, the focus regarding SmartViz will be given to the differences and additions from an UI perspective introduced in R2. For further details regarding the toolkit please refer to Section 2.3 in D4.6[43].

4.7.1.2 Internal Operation & Technologies

The DMT is designed to handle various types of data from different workflows and pipelines. This data can have different characteristics such as velocity, format, and type, and it can be delivered through live or batch streams. SmartViz, as part of the toolkit, accepts and transforms the input data into meaningful visualisations based on their nature and format. The primary goal of the SmartViz toolkit is to effectively support and accommodate all these different data scenarios while meeting the needs of end-users and facilitating decision-making processes.

The internal architecture and underlying technologies of SmartViz have largely remained the same at a high-level perspective since R1. However, significant enhancements have been made in R2 to introduce new visualisation widgets and dashboards for the new use cases. The R1 use cases have also been enriched with filters and the support of new component visualisations, based on the valuable user feedback.

To facilitate the enhancements along with the new visualisations, SmartViz has established connections with additional components beyond what was available in R1. These connections are showcased within the DMT dashboards. To support these connections, the necessary mechanisms were developed, and new libraries written in Typescript and JavaScript were integrated into the toolkit. These additions have expanded the range of visualisation widgets available. For further details regarding SmartViz please refer to Section 3.2 in D4.6.

4.7.1.3 Role in R2 and associated use cases

SmartViz serves as the final component in the MARVEL pipeline and plays a crucial role in providing meaningful insights and interactive capabilities to end-users, thereby facilitating urban planning, safety, and other objectives outlined by the MARVEL pilots.

SmartViz was deployed as well in the R1 and served as the UI of the DMT catering to the visualisation requirements of all R1 use cases and user scenarios specified by the pilots. Since

[43] D4.6 - MARVEL's decision-making toolkit – final version," Project MARVEL, 2023. https://doi.org/10.5281/zenodo.8147077.

the R1 release, more visualisation widgets have been developed and released to satisfy the specified user journeys and needs, but also to properly represent the incoming data. The visualisation widgets are complemented by a range of filtering options which have been expanded and enhanced in R2.

SmartViz is utilised across all R2 use cases and it carries out the visualisation needs of all the undertaken use cases that are described in Section 2. The pool of visualisation widgets includes simple charts and graphs, complex timeline representations, geospatial depictions, audio and visual representation of historical and real-time feeds, and real-time rendering of data. For the final version, we have completed the widget pool and selected appropriate widgets that align with the user requirements and the respective data. Since R1, there have been no changes in the deployment of SmartViz which remains at the cloud. For further details regarding SmartViz please refer to Section 2.3 in D4.6

4.7.2 MARVEL Data Corpus-as-a-Service

This subsection briefly presents the MARVEL Data Corpus, its main technologies, and its use under R2.

4.7.2.1 Overview

The aim of the MARVEL Data Corpus is to provide Machine Learning datasets to the relevant scientific and industrial communities. It obtains anonymised and annotated video and audio data from the MARVEL pilots and stores it in the Corpus. Besides offering data to the external communities, the Corpus can also be utilised by MARVEL's internal operations by allowing its Cloud/back-end's Artificial Intelligence elements to access the datasets and utilise them to train and assess ML models.

4.7.2.2 Internal Operation & Technologies

Several technologies have been employed in the creation of the Corpus. These include server-side technologies for storing the core data, client applications that allow other users or components to interface with the Corpus, such as with graphical user interfaces (GUIs), along with a series of REST APIs that can be used ad-hoc by the relevant users. A native installation of Hadoop Distributed File System (HDFS)[44] among the Corpus nodes is responsible for saving the data files while it keeps multiple copies of data across low-cost machines, thereby providing a large bandwidth for accessing them and increased resilience to hardware failures. In addition, HBase[45], an open-source, non-relational, distributed database, takes care of managing the relationship of the data based on a non-schema less approach. Zookeeper is the administrative application on which the HBase distributed database is based. This module is open-source and provides management services such as keeping track of configurations, providing naming and distributed synchronisation. Accessing these services can be done through Ambar's web interface which provides an uncomplicated Hadoop management solution with a webpage and RESTful APIs.

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

[45] https://hbase.apache.org

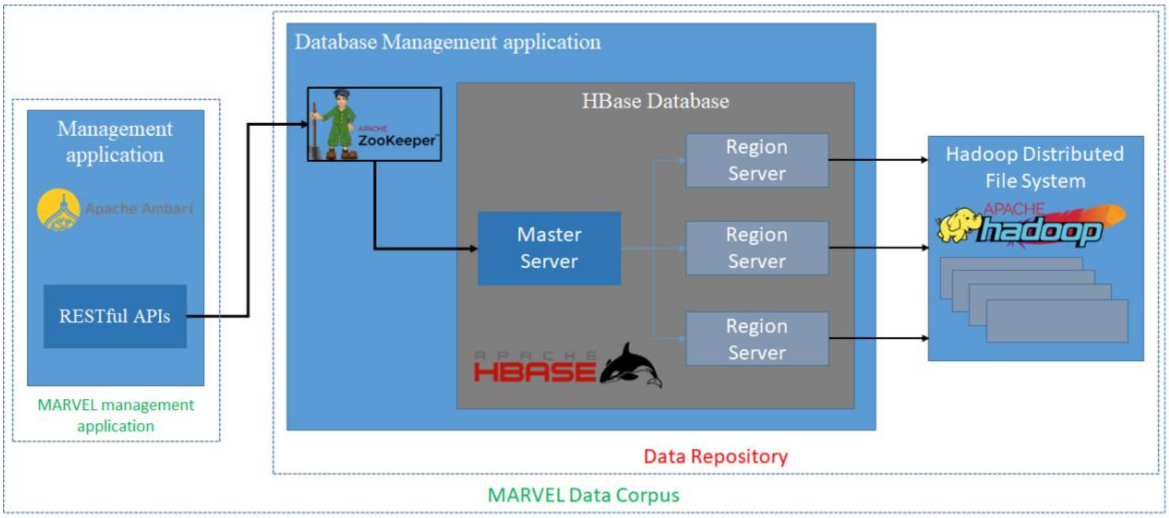


Figure 44: Internal architecture for the core storage of the MARVEL Data Corpus

Furthermore, applications in JAVA have been developed that permit the server-side APIs to be called programmatically and interact with the repository. One example is an application that can move files from a folder and put them into the Corpus, as well as a program that links specified Kafka topics and ingests the inference results released from the MARVEL AI components.

Finally, two GUIs have been implemented providing two different levels of access to the end user of the Corpus. The first one, which requires user authentication, includes all the datasets and snippets that have been injected to the Data Corpus, providing full access to them in terms of update and delete actions. The second one, which is publicly available and does not require any user authentication, includes datasets that have been marked as "complete and public" while the users cannot perform update or delete actions to the relative records. In both GUIs, there is the capability of performing simple and advanced queries to the stored datasets while both interfaces provide the capability of downloading the datasets/snippets locally.

Figure 45: The private and the public GUIS of the Data Corpus

Underneath, an Elastic Search-Logstash-Kibana (ELK) stack has been deployed in order to monitor the requests of the exposed REST APIs along with the overall status of the Data Corpus.

4.7.2.3 Role in R2 and associated use cases

The purpose of the Corpus is threefold. Firstly, it stores anonymised and annotated datasets from the pilots. Interfaces have been created to make this process easier, and these consist of both programmatic and graphical elements. Additionally, the Corpus can automatically ingest data, such as video/audio, which have been sent by the StreamHandler component, as well as inference results which are published by the DFB Kafka topics.

As a result, the internal MARVEL elements in the cloud can obtain and use these sets of data. AI parts can access the relevant information and train their ML models. For more information about the ML processes being carried out, please look at the subsections related to these components. Additionally, the datasets can either be obtained programmatically or downloaded through the graphical user interface.

In conclusion, the Corpus makes available the ingested datasets to external users (both scientific and industrial) in an accessible way. Currently, there are multiple datasets from the pilots available to be accessed. As the project progresses, the amount of this data is expected to increase as more data from pilots are anonymised and labelled.

5 R2 Design and Specifications

This section presents the results of the integration and deployment activities for the final version of the MARVEL Integrated framework (R2), i.e., the final design and specifications of the R2 release. Initially, the ‘**AI Inference pipeline**’ **reference architecture** is presented along with its design rationale. This reference architecture is the **backbone of the R2 framework** and drove the architecture instantiations in all R2 use cases. Subsequently, Sections 5.2 and 5.3 present the implemented **I/O interfaces - APIs** and **Data Models** respectively. These form the **two main integration pillars** of the MARVEL framework. Finally, the **UI/UX approach** that was adopted is described, followed by a presentation of the **infrastructure** that was used for the R2 release.

5.1 ‘AI Inference pipeline’ reference architecture

The MARVEL conceptual architecture, presented in Figure 1 of D1.3, was used to steer the design of the specific architectural configurations for addressing the use cases selected for R2. For completeness, a revised version of the MARVEL conceptual architecture is also presented below in Figure 46. Details of the revisions of the MARVEL conceptual architecture since D1.3 (M08) are reported in D6.1.

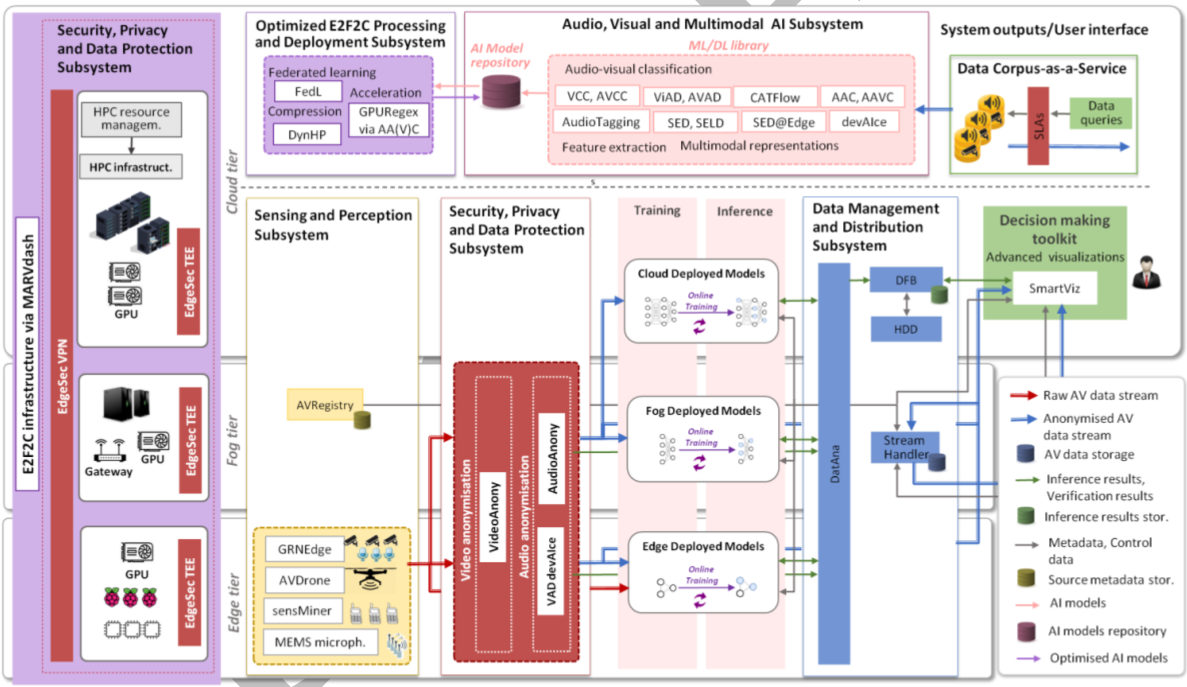


Figure 46: MARVEL conceptual architecture

Furthermore, the experience gained from the MVP (D5.1 *‘MARVEL Minimum Viable Product’*) and from R1 (D5.4 *‘MARVEL Integrated framework – initial version’*) was leveraged for the design of R2, which built upon these previous outcomes.

As already explained in Section 3.2, the R2 design process started with the analysis of the use cases that R2 would address (Section 2.2). This process was complemented by simultaneous activities for analysing the available infrastructure at the three pilot sites and provisional mappings of the MARVEL components to the use cases and infrastructure nodes. In parallel, the I/O interfaces and data models, as well as UI/UX aspects, were elaborated and considered in the investigations of architectural configuration options. Another factor that played a crucial role in the architectural design was the attempt to consolidate as much as possible the necessary I/O interfaces and data models and streamline them into unified interface and data model types that could serve the needs of multiple components in order to reduce integration complexity. During these activities, tentative architectural diagrams were prepared to illustrate the various possible options that were being considered.

Common functional requirements. The analysis of the use cases that were to be addressed by R2 led to the determination of a common theme that referred to the need of analysing source multimodal audio-visual (AV) data to extract meaningful inference results and deliver them to the user. More specifically, the common functional requirements that were identified are:

- Access live feeds of AV sources.
- Anonymise streamed AV data as close to the source as possible.
- Provide live AV data to AI components for analysis so that they can produce live inference results.
- Collect the inference results from all AI components.
- Homogenise the inference results.
- Persistently store the inference results.
- Provide real-time and historical inference results to the user with dedicated visualisations.
- Persistently store AV data.
- Provide real-time and historical AV data to the user.

The analysis of this set of common requirements led to the conclusion that a universal reference architecture scheme was required that could be applied to all R2 use cases with suitable adaptations on a case-by-case basis. This reference architecture was the outcome of the distillation of all requirements and ensures that there is a consistent and coherent approach in all applications of MARVEL for extracting and delivering inference results from the analysis of multimodal AV data. Following multiple iterations and revisions, the MARVEL Consortium produced such a reference architecture, which is also referred to as the 'AI Inference Pipeline', illustrated in Figure 47 below.

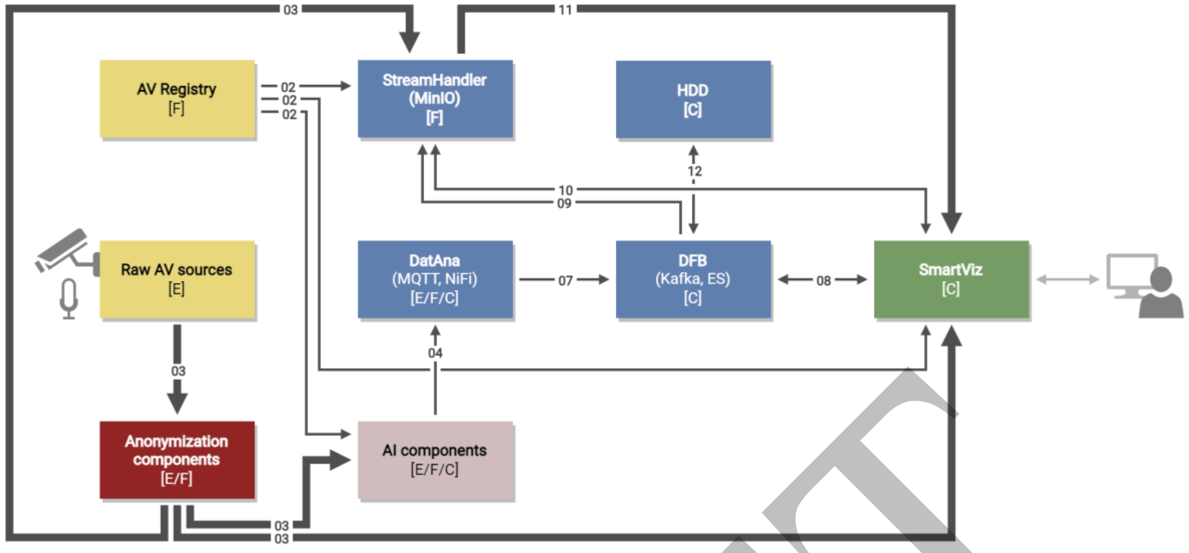


Figure 47: MARVEL 'AI Inference Pipeline' reference architecture diagram

The 'AI Inference Pipeline' reference architecture diagram contains abstractions for certain components with similar functionalities that are grouped together. More specifically, the Anonymisation components refer to the AudioAnony, VAD, and VideoAnony components and the AI components refer to the CATFlow, TAD, VAD, ViAD, AVAD, VCC, AVCC, AT, SED, AAC, YOLO-SED, RBAD, and SELD components. A full description of all components referenced in the 'AI Inference Pipeline' is provided in Section 4.

The main interactions between the components in the 'AI Inference Pipeline' can be described as follows:

- **Anonymisation components** receive AV data streams from raw **AV sources** (cameras and microphones). [interface #03].
- **AI components** request the metadata of available AV sources (including anonymisation components) from the **AV Registry**. [interface #02].
- **AI components** receive anonymised AV data streams from the **Anonymisation components**. [interface #03].
- **AI components** publish their raw inference results to dedicated topics on the **DatAna** MQTT broker which is hosted on the same layer (E/F/C) as the AI components. [interface #04]. A DatAna NiFi node receives the raw inference results from the DatAna MQTT broker, which is hosted on the same layer (E/F/C) as the DatAna NiFi node. DatAna NiFi nodes transform the raw inference results they collect to SDM-compliant inference results and push them to a DatAna NiFi node at a higher layer (DatAna NiFi Edge pushes results to DatAna NiFi Fog and DatAna NiFi Fog pushes results to DatAna NiFi Cloud).
- **DatAna NiFi** (Cloud node) publishes the SDM-compliant inference results it collects to a topic on a Kafka broker of the **DFB**. The DFB persistently stores the received results on an Elastic Search (ES) database. [interface #07].
- **SmartViz** accesses the incoming inference results at the **DFB** in real time by subscribing to the Kafka topics where they are published by DatAna. SmartViz also accesses the historical inference result data stored in the ES database of the DFB. SmartViz displays the inference results to the user through appropriate visualisations.

SmartViz also allows the user to verify the displayed inference results and this verification information is fed back to the DFB by being published to a dedicated Kafka topic. [interface #08]

In parallel, the following interactions take place for SmartViz to gain access to AV data:

- **SmartViz** and **StreamHandler** request the metadata of available AV sources (including anonymisation components) from the **AV Registry**. [interface #02].
- **StreamHandler** receives anonymised AV data streams from the **Anonymisation components**. [interface #03] StreamHandler segments and stores the anonymised AV data in a MinIO repository for subsequent access.
- **StreamHandler** accesses the incoming inference results at the **DFB** in real time by subscribing to the Kafka topics where they are published by DatAna [interface #09] to produce AV files that correspond to these results.
- **SmartViz** requests and receives AV data from **StreamHandler** that refer to a particular inference result. [interfaces #10, #11].
- **SmartViz** receives an anonymised AV data stream from an **Anonymisation component** upon user demand. [interface #03] The connection is not continuous and is initiated only when a user requests a live AV feed from the location where an event has previously been detected.

The ‘AI Inference Pipeline’ operation is also complemented by the following interaction that serves the purposes of performance optimisation:

- The **HDD** receives the current Kafka topic partition configuration from the **DFB** and returns a recommendation for an optimised Kafka topic partition configuration. [interface #12].

In terms of component mapping to the E/F/C layers, the ‘AI Inference Pipeline’ architecture proposes specific deployment layers for certain components but also incorporates an inherent flexibility as far as other components are concerned. The following aspects were considered for the determination of the possible variations of the E/F/C layer to which each component could be deployed:

- Use case requirements;
- Nature of the component;
- Needs for direct interaction with other components;
- Data privacy and security issues;
- Bandwidth preservation aspects;
- Performance and accuracy aspects;
- Available infrastructure.

For some of the components, the possibilities of deployment layer were more limited due to their nature. Specifically, these were:

- **AV sources** (microphones, cameras). Due to their nature of capturing data from the real world, AV sources are located at the Edge.
- **DatAna**. Due to its pervasive nature, DatAna is foreseen to span across all three layers (Edge, Fog, and Cloud). Separate DataAna MQTT brokers and DatAna NiFi nodes can

reside on any infrastructure node at any layer, if an AI component that produces inference results also resides there.

Regarding the other components that are present in the ‘AI Inference Pipeline’, the deployment possibilities were determined by considering the aforementioned aspects. Specifically regarding (i) Data privacy and security, (ii) Bandwidth preservation, and (iii) Performance and security, the impact that the E2F2C deployment layer of a component would have on these three aspects was assessed. In this impact assessment analysis, components were considered in groups of components with similar roles. A rating scheme (Low, Medium, High) was assigned to the impact of each deployment option for each component group towards the three considered aspects. In this sense, it was assumed that the Edge layer provides a high degree of data privacy and security and bandwidth preservation, as data can be processed close to their source, but also a low performance and accuracy (e.g., for AI components), as the infrastructure at the Edge typically has lower computational resources. The Cloud layer provides a low degree of data privacy and security and bandwidth preservation, as data have to be transferred from the Edge to the Cloud, but also a high performance and accuracy (e.g., for AI components), as the infrastructure at the Cloud (e.g., HPC) can provide abundant computational resources. The Fog layer was seen as a middle ground, providing increased data privacy and security and bandwidth preservation compared to the Cloud layer and increased performance and accuracy compared to the Edge layer. Some options for the deployment layer were ruled out as unsuitable for certain component groups. The ratings that were assigned to the deployment options guided the decisions on the EFC layers that would be suitable for the deployment of each component group along with the remaining aspects under consideration and especially the infrastructure that was available at the MARVEL pilots.

The following Table 5 presents the aforementioned impact assessment analysis that was performed as well as the possible deployment layer options that were eventually promoted for each R2 component group.

Table 5: Deployment option prioritisation for components in the 'AI Inference Pipeline'

Deployment options					
Component Group	**Considered aspects**	**Impact rating by EFC layer**			**Promoted R2 Deployment layer options**
		Edge	Fog	Cloud	
Anonymisation Components (AudioAnony, VAD, VideoAnony)	Data Privacy / Security	HIGH	MEDIUM	N/A	Edge / Fog
	Bandwidth preservation	HIGH	MEDIUM	N/A	
	Performance / Accuracy	LOW	MEDIUM	N/A	
AI Components (CATFlow, TAD, VAD, ViAD, AVAD, VCC, AVCC, AT, SED, AAC, GPURegex, YOLO-SED, RBAD, SELD)	Data Privacy / Security	HIGH	MEDIUM	LOW	Edge / Fog / Cloud
	Bandwidth preservation	HIGH	MEDIUM	LOW	
	Performance / Accuracy	LOW	MEDIUM	HIGH	
AV Management Components (AV Registry, StreamHandler)	Data Privacy / Security	N/A	HIGH	LOW	Fog
	Bandwidth preservation	N/A	HIGH	LOW	
	Performance / Accuracy	N/A	MEDIUM	HIGH	
Inference result storage and visualisation (DFB, SmartViz, HDD)	Data Privacy / Security	N/A	HIGH	MEDIUM	Cloud
	Bandwidth preservation	N/A	MEDIUM	LOW	
	Performance / Accuracy	N/A	LOW	HIGH	

The 'AI Inference Pipeline' reference architecture was used as a basis to specify the exact architecture instantiation for each use case. The specific configuration of each instantiation was determined by the following factors:

Use case requirements, as distilled from the user journeys described in D4.6, but also functional and non-functional requirements from D2.1 and their refinements in D6.1 and D6.3.

AV sources that are planned or need to be employed by the use case.

Anonymisation components that would need to be involved, depending on the data modalities present in the use case AV sources (audio, video).

AI components that would need to be involved to address the specific (AI) tasks of the use case.

Available infrastructure at each pilot for hosting the MARVEL components.

The design choices that were made for the architecture that was implemented for each use case were also affected by an attempt to maximise the representation of individual MARVEL components and to demonstrate a variance between the different configurations to validate the capability ofMARVEL to conform to the needs of diverse use cases.

In R2, several steps were taken towards shifting the positioning of anonymisation and AI models as close to the Edge as possible. To that end, additional edge devices were secured in all pilots, including NVidia Jetson, Raspberry Pi and PC devices. In addition, the fog server of GRN was significantly upgraded, allowing it to host AI models that would otherwise be positioned at the cloud. These additional resources were considered when designing the five new use cases that were introduced for R2 (GRN1, GRN2, MT2, MT4, UNS2) and allowed to host anonymisation and AI models closer to the edge. Furthermore, the architecture for some use cases that had been applied in R1 (GRN3, GRN4), was partially revised due to the availability of the new infrastructure and allowed to shift some AI and anonymisation components to the edge and fog layers.

Nevertheless, limitations were still present in the infrastructure at the edge and fog layers and an efficient allocation of the available resources was required to accommodate the needs of all ten R2 use cases. GPU acceleration was enabled for the inference calculations in most of the AI models so that the corresponding resources on the host devices could be fully exploited. Wherever it was not possible to host all AI components at the edge and fog layers, the cloud layer was used, where resources were more abundantly available.

Furthermore, in some cases, the architecture configuration and component deployment locations were also largely affected by imposed network security restrictions and data privacy issues. Such was the case in the MT pilot (MT1, MT2, MT3 and MT4 use cases), where customised solutions needed to be devised to fit the exact use case requirements.

The next sub-section 5.4 presents the specific instantiations of the MARVEL 'AI Inference Pipeline' architecture that were implemented for the use cases addressed by R2. Each sub-section presents a table with the exact components that were applied and the infrastructure nodes on which they were deployed. Each sub-section also presents a diagram of the respective system architecture that also incorporates information related to deployment, as it depicts the exact component instances (services) that were deployed in relation to the infrastructure nodes.

The diagrams also feature a unified annotation with labels that refers to the consolidated I/O interface types that were implemented (described in detail in Section 5.2). Thin arrows represent the exchange of text-based data and thick arrows represent the exchange of AV (binary) data.

Finally, it is noted that the architecture that was specified for AI Training and the operation of the Data Corpus are presented in separate sub-sections (Sections 5.4.11 and 5.4.12 respectively), as these were applicable to all R2 use cases.

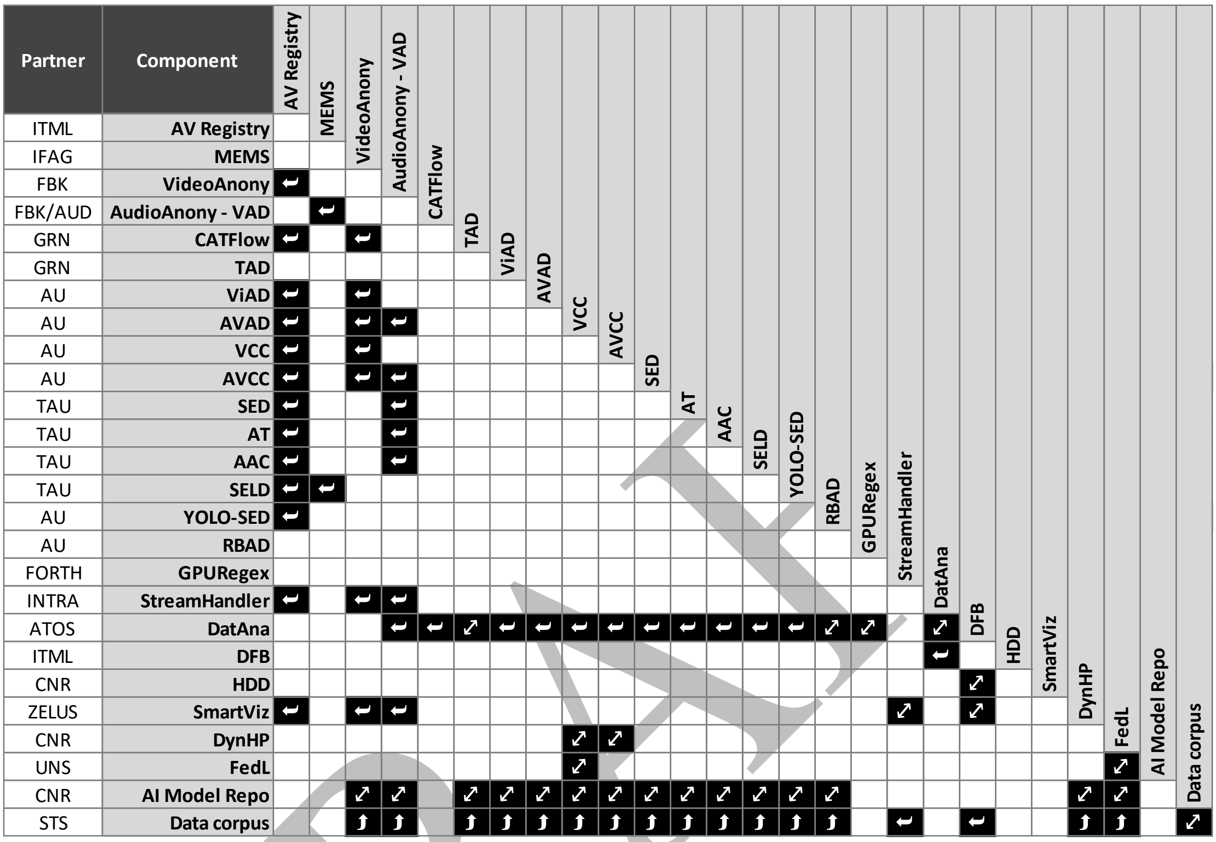
5.2 I/O Interfaces - APIs

The specification of I/O Interfaces and Data Models are the two main pillars in the integration of multiple components into a unified system to ensure its successful operation. This section presents the first pillar, i.e., the specification of I/O Interfaces – APIs.

During the design of the reference inference pipeline architecture and the specific implementation of the R1 and R2 use cases, the I/O interfaces that would be required to support the communication between individual components were considered. The strategy for the specification of the I/O interfaces and APIs was based on the following objectives:

- Distil the data exchange requirements of the MARVEL components to consolidate the necessary I/O interfaces as much as possible and consequently reduce integration complexity.
- Decouple as much as possible the direct data exchange between pairs of individual component instances to reduce integration complexity, i.e., avoid the use of REST APIs wherever possible and promote the use of pub/sub distributed messaging systems.
- Implement open, industry-standard approaches for increased interoperability, scalability and expandability.

Following the analysis of the data exchange requirements of individual components, an attempt was made to map the pairwise communication needs between individual components. This process led to the identification of similarities between the requirements of different components, which were leveraged in order to streamline the various pairwise communication needs into universal I/O interface types that could be shared between groups of components and implemented across the entire MARVEL framework. Figure 48 illustrates a connectivity matrix that was used to map out the pairwise communication needs between the individual MARVEL components, including information on the directionality of each identified pairwise communication need.



Partner	Component	AV Registry	MEMS	VideoAnony	AudioAnony - VAD	CATFlow	TAD	ViAD	AVAD	VCC	AVCC	SED	AT	AAC	SELD	YOLO-SED	RBAD	GPURegex	StreamHandler	DatAna	DFB	HDD	SmartViz	DynHP	FedL	AI Model Repo	Data corpus
ITML	AV Registry																										
IFAG	MEMS																										
FBK	VideoAnony	↵																									
FBK/AUD	AudioAnony - VAD		↵																								
GRN	CATFlow	↵		↵																							
GRN	TAD																										
AU	ViAD	↵		↵																							
AU	AVAD	↵		↵	↵																						
AU	VCC	↵		↵																							
AU	AVCC	↵		↵	↵																						
TAU	SED	↵			↵																						
TAU	AT	↵			↵																						
TAU	AAC	↵			↵																						
TAU	SELD	↵	↵																								
AU	YOLO-SED	↵																									
AU	RBAD																										
FORTH	GPURegex																										
INTRA	StreamHandler	↵		↵	↵																						
ATOS	DatAna				↵	↵	⤢	↵	↵	↵	↵	↵	↵	↵	↵	↵	⤢	⤢		⤢							
ITML	DFB																			↵							
CNR	HDD																				⤢						
ZELUS	SmartViz	↵		↵	↵														⤢		⤢						
CNR	DynHP									⤢	⤢																
UNS	FedL									⤢															⤢		
CNR	AI Model Repo			⤢	⤢		⤢	⤢	⤢	⤢	⤢	⤢	⤢	⤢	⤢	⤢	⤢							⤢	⤢		
STS	Data corpus			⤴	⤴		⤴	⤴	⤴	⤴	⤴	⤴	⤴	⤴	⤴	⤴	⤴		↵		↵			⤴	⤴		⤢

Figure 48: MARVEL component connectivity matrix for I/O interface specification, including indication of bidirectional/unidirectional connections and directionality of unidirectional connections

Meanwhile, the foreseen pairwise interactions between MARVEL components were mapped to the MARVEL use cases by means of a matrix depicted in Figure 49. This tabular organisation of the information assisted in distilling the possible I/O Interface types and consolidating them to ones that could serve the needs of multiple components.

Component A	Direction	Component B	I/O Interface Type	GRN3: Traffic Anomalous Events	GRN4: Trajectories	MT1: Crowd Monitoring	MT3: Parking Lot	UNS1: Drone Crowd Classification	AI Training	Data Corpus Data Aggregation
AV Registry	→	CATFlow	2	Y	Y	Y	N	N	N	N
AV Registry	→	ViAD	2	N	N	Y	N	N	N	N
AV Registry	→	AVAD	2	Y	N	N	Y	N	N	N
AV Registry	→	VCC	2	N	N	Y	N	Y	N	N
AV Registry	→	AVCC	2	N	Y	N	N	N	N	N
AV Registry	→	SED	2	N	Y	N	Y	N	N	N
AV Registry	→	AT	2	Y	N	N	Y	N	N	N
AV Registry	→	StreamHandler	2	Y	Y	Y	Y	Y	N	N
AV Registry	→	SmartViz	2	Y	Y	Y	Y	Y	N	N
MEMS	→	AudioAnony - VAD	1	N	N	N	Y	Y	N	N
VideoAnony	→	CATFlow	3	Y	Y	Y	N	N	N	N
VideoAnony	→	ViAD	3	N	N	Y	N	N	N	N
VideoAnony	→	AVAD	3	Y	N	N	Y	N	N	N
VideoAnony	→	VCC	3	N	N	Y	N	Y	N	N
VideoAnony	→	AVCC	3	N	Y	N	N	N	N	N
VideoAnony	→	SED	3	N	Y	N	N	N	N	N
VideoAnony	→	AT	3	Y	N	N	N	N	N	N
VideoAnony	→	StreamHandler	3	Y	Y	Y	Y	Y	N	N
VideoAnony	←→	AI Model Repository	13	N	N	N	N	N	Y	N
VideoAnony	←	Data corpus	14	N	N	N	N	N	Y	N
AudioAnony - VAD	→	AVAD	3	N	N	N	Y	N	N	N
AudioAnony - VAD	→	AVCC	3	N	N	N	N	N	N	N
AudioAnony - VAD	→	SED	3	N	N	N	Y	N	N	N
AudioAnony - VAD	→	AT	3	N	N	N	Y	N	N	N
AudioAnony - VAD	→	StreamHandler	3	N	N	N	Y	Y	N	N
AudioAnony - VAD	→	SmartViz	3	N	N	N	Y	Y	N	N
AudioAnony	←→	AI Model Repository	13	N	N	N	N	N	Y	N
AudioAnony	←	Data corpus	14	N	N	N	N	N	Y	N
CATFlow	→	DatAna	4	Y	Y	Y	Y	N	N	N
TAD	→	DatAna	4	Y	Y	N	N	N	N	N
TAD	←	DatAna	5	Y	Y	N	N	N	N	N
TAD	←→	AI Model Repository	13	N	N	N	N	N	Y	N
TAD	←	Data corpus	14	N	N	N	N	N	Y	N
ViAD	→	DatAna	4	N	N	Y	N	N	N	N
ViAD	←→	AI Model Repository	13	N	N	N	N	N	Y	N
ViAD	←	Data corpus	14	N	N	N	N	N	Y	N
AVAD	→	DatAna	4	Y	N	N	Y	N	N	N
AVAD	←→	AI Model Repository	13	N	N	N	N	N	Y	N
AVAD	←	Data corpus	14	N	N	N	N	N	Y	N
VCC	→	DatAna	4	N	N	Y	N	Y	N	N
VCC	←→	AI Model Repository	13	N	N	N	N	N	Y	N
VCC	←	Data corpus	14	N	N	N	N	N	Y	N
AVCC	→	DatAna	4	N	Y	N	N	N	N	N
AVCC	←→	AI Model Repository	13	N	N	N	N	N	Y	N
AVCC	←	Data corpus	14	N	N	N	N	N	Y	N
SED	→	DatAna	4	N	Y	N	Y	N	N	N
SED	←→	AI Model Repository	13	N	N	N	N	N	Y	N
SED	←	Data corpus	14	N	N	N	N	N	Y	N
AT	→	DatAna	4	Y	N	N	Y	N	N	N
AT	←→	AI Model Repository	13	N	N	N	N	N	Y	N
AT	←	Data corpus	14	N	N	N	N	N	Y	N
StreamHandler	←→	SmartViz	10	Y	Y	Y	Y	Y	N	N
StreamHandler	→	Data corpus	11	Y	Y	Y	Y	Y	N	Y
DatAna	→	DFB	7	Y	Y	Y	Y	Y	N	N
DFB	←→	HDD	12	Y	Y	Y	Y	Y	N	N
DFB	←→	SmartViz	8	Y	Y	Y	Y	Y	N	N
DFB	→	Data corpus	9	Y	Y	Y	Y	Y	N	Y
DynHP	←→	AI Model Repository	13	N	N	N	N	N	Y	N
DynHP	←	Data corpus	14	N	N	N	N	N	Y	N
FedL VCC Server	←	FedL VCC client	15	N	N	N	N	N	Y	N
FedL	←→	AI Model Repository	13	N	N	N	N	N	Y	N

Figure 49: MARVEL component pairwise interaction mapping to I/O Interface Types and architectures

This mapping and associated analysis gave way to the determination of 16 MARVEL I/O interface types that were used universally in the MARVEL integrated framework. They are presented in **Table 6** and elaborated in more detail in the following Sections 5.2.1 - 5.2.16. They are divided in 3 categories:

- I/O Interface Types #01 - #12: Applied in the ‘AI Inference Pipeline’ architecture for the 10 R2 use cases (Sections 5.2.1 - 5.2.12)
- I/O Interface Types #13 - #15: Applied in the ‘AI Training’ architecture (Section 5.2.13 - 5.2.15)
- I/O Interface Types #11 and #16: Applied in the Data Corpus Data Aggregation architecture (Sections 5.2.11, 5.2.16)

The numerical indices used to number the I/O interface types in **Table 6** are also used in every graphic depiction of architecture diagrams as annotation labels for describing the interactions between the components included in each diagram.

Table 6: MARVEL R2 I/O Interface Types

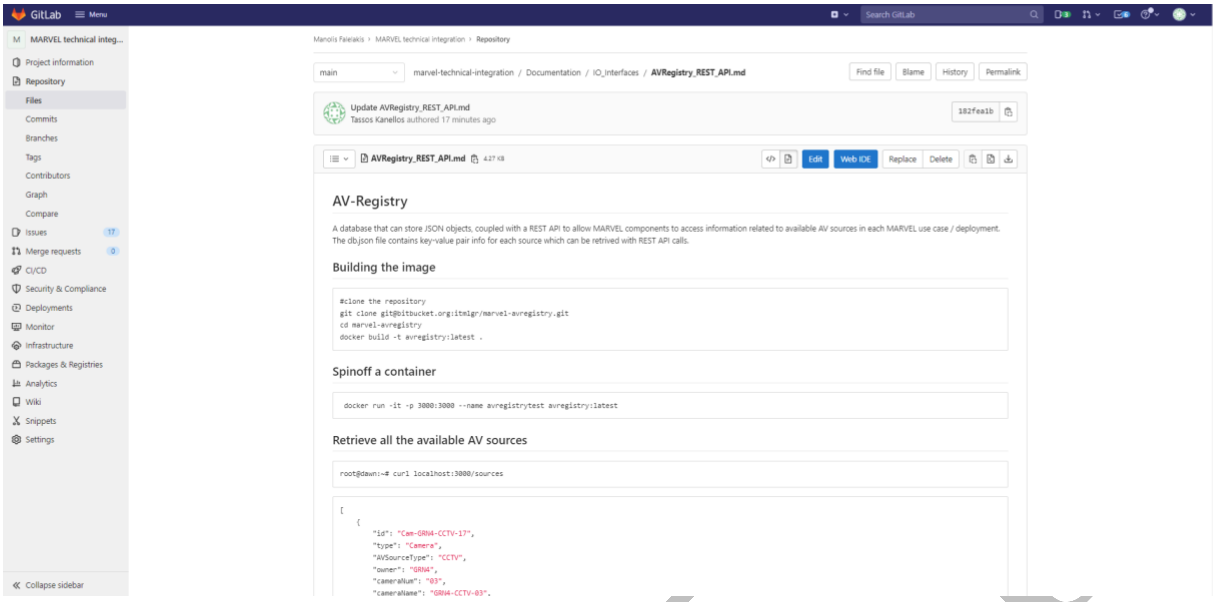
Index	I/O Interface Type title	I/O Interface Type description	API/Protocol	Relevant components	Relevant Data Models
1	**Onboard AV source access** (Inference Pipeline)	A component (AudioAnony-VAD, VideoAnony, SELD) accesses an AV source that is hosted on the same device (e.g., MEMS microphone, GoPro camera)	OS driver	MEMS, AudioAnony-VAD, VideoAnony, SELD	N/A
2	**AV Registry access** (Inference Pipeline)	A component (CATFlow, ViAD, AVAD, VCC, AVCC, SED, AT, AAC, StreamHandler, SmartViz) accesses the AV Registry REST API to receive metadata for a specific AV source id. The metadata include the AV source stream URL that can be used to access the AV data stream (interface type #03).	REST API	AV Registry, CATFlow, ViAD, AVAD, VCC, AVCC, SED, AT, AAC StreamHandler, SmartViz	Camera entity
3	**AV streaming** (Inference Pipeline)	A component (VideoAnony, AudioAnony-VAD, CATFlow, ViAD, AVAD, VCC, AVCC, SED, AT, AAC, YOLO-SED, StreamHandler, SmartViz) receives the AV stream from an AV source (CCTV IP Camera, VideoAnony, AudioAnony-VAD) using RTSP.	RTSP	VideoAnony, AudioAnony-VAD, CATFlow, ViAD, AVAD, VCC, AVCC, SED, AT, AAC, YOLO-SED, StreamHandler, SmartViz	Camera entity
4	**DatAna: AI inference result publication** (Inference Pipeline)	An AI component (CATFlow, TAD, VAD, ViAD, AVAD, VCC, AVCC, SED, AT, AAC, YOLO-SED, RBAD, GPURegex, SELD) publishes inference results to a dedicated topic on a DatAna MQTT broker that resides on the same infrastructure node. A DatAna NiFi node on the same infrastructure node receives the inference results by subscribing to the same topic on the DatAna MQTT broker.	MQTT publish, MQTT subscribe	DatAna MQTT, CATFlow, TAD, VAD, ViAD, AVAD, VCC, AVCC, SED, AT, AAC, YOLO-SED, RBAD, GPURegex, SELD	each AI component uses a different data model to structure its own raw inference results
5	**DatAna: AI inference result consumption**	A MARVEL component consumes raw inference results produced by an AI component in real time by subscribing to	MQTT subscribe	DatAna MQTT, CATFlow, TAD, RBAD, AAC,	CATFlow, AAC and YOLO-SED

	(Inference Pipeline)	the topic of the DatAna MQTT broker, where the AI component publishes its inference results.		GPURegex, Arduino Proxy (GRN1)	raw inference result data models
6	**DatAna Inter-Agent Communication** (Inference Pipeline)	DataAna NiFi inter-agent communication to relay inference results from NiFi nodes on lower EFC layers to NiFi nodes on higher EFC layers: (a) DatAna NiFi Edge forwards information to DatAna NiFi Fog (b) DataAna NiFi Fog forwards information to DatAna NiFi Cloud	NiFi communication	DatAna NiFi	N/A (NiFi Site-to-Site communication, enabled via configuration files and the NiFi UI data flows)
7	**DatAna – DFB interaction** (Inference Pipeline)	DataAna NiFi Cloud publishes transformed inference results (SDM-compliant) to the DFB Kafka brokers for persistent storage. Each inference result is published to a dedicated Kafka topic according to the AI component that produced the original result.	Kafka publish	DatAna, DFB	Alert, Anomaly, MediaEvent entities
8	**DFB – SmartViz interaction** (Inference Pipeline)	(a) SmartViz receives inference results in real time by subscribing to the dedicated DFB Kafka topics where transformed (SDM-compliant) AI inference results are being published by DatAna. (b) SmartViz accesses historical inference results from the Elastic Search (ES) database using a REST API (DFB ES-Proxy service) (c) When a user of SmartViz verifies an inference result, SmartViz publishes a message to a dedicated topic on the DFB Kafka brokers with the verification information. The DFB updates the corresponding inference result entry accordingly in the ES database.	(a) Kafka subscribe, (b) REST API, (c) Kafka publish	DFB, SmartViz	Alert, Anomaly, MediaEvent, InferenceVerification entities
9	**DFB – StreamHandler interaction** (Inference Pipeline)	StreamHandler receives inference results in real time by subscribing to the dedicated DFB Kafka topics where transformed (SDM-compliant) AI inference results are being published by DatAna.	Kafka subscribe	DFB, StreamHandler	Alert, Anomaly, MediaEvent entities
10	**StreamHandler – SmartViz interaction** (Inference Pipeline)	SmartViz requests AV data from StreamHandler by providing the id of an inference result. StreamHandler returns the filename of the requested AV data.	REST API	StreamHandler, SmartViz	N/A
11	**StreamHandler MinIO AV Data access** (Inference Pipeline, Data Corpus Data Aggregation)	SmartViz and Data Corpus receive AV data from StreamHandler by accessing the files stored in its MinIO database.	DB I/O	StreamHandler, SmartViz, Data Corpus	N/A

12	**DFB – HDD** (Inference Pipeline)	HDD receives the current Kafka topic partition configuration from the DFB and returns a recommendation for an optimised Kafka topic partition configuration.	REST API	DFB, HDD	Kafka Topic Partition Allocation
13	**AI Model Repository** (AI Training)	DynHP, FedL and AI components store and retrieve ML models in the Cloud-based AI Model Repository based on MinIO.	DB I/O	AudioAnony, VideoAnony, TAD, SED, AT, ViAD, AVAD, VCC, AVCC, DynHP, FedL	MLModel entity
14	**AV Dataset reception** (AI Training)	An AI component (training mode) receives annotated AV data from the Data Corpus or from a local repository and performs ML/DL training with the received dataset.	REST API, DB I/O	DataCorpus, AudioAnony, VideoAnony, TAD, SED, AT, ViAD, AVAD, VCC, AVCC, DynHP	N/A
15	**FedL VCC Server - FedL VCC Client** (AI Training)	The FedL VCC server exchanges information bidirectionally with the FedL VCC client to process a compressed/optimised AI model and update it after performing federated learning.	Federated Learning API (based on gRPC)	FedL, VCC	None, internal raw communication model
16	**DFB – DataCorpus** (Data Corpus Data Aggregation)	(a) The Data Corpus receives inference results in real time by subscribing to the dedicated DFB Kafka topics where transformed (SDM-compliant) AI inference results are being published by DatAna. (b) The Data Corpus also receives user verification information about specific inference results by subscribing to the dedicated DFB Kafka topic, where they are published by SmartViz.	Kafka subscribe	DFB, DataCorpus	Alert, Anomaly, MediaEvent, InferenceVerification entities

The I/O Interface type protocols that were selected were consistent with the overall objectives guiding the design of the interfaces. In particular, the DatAna and DFB components implement a distributed messaging system (MQTT and Kafka respectively). These publication / subscription messaging systems based on brokers are now considered to be industry standards with multiple advantages. Most importantly, they allow the decoupling of direct communication between pairs of components and enabling a common interface to be used by multiple endpoints. In other cases (e.g., AV Registry, StreamHandler, DFB ES-Proxy, DFB-HDD, Data Corpus), RESTful APIs have been implemented, conforming to the best practices followed by the industry. In some cases, direct IO operations in data repositories have been implemented, in conjunction with specific naming and information organisation conventions. Such are the cases of the MinIO data store implemented by the StreamHandler and the AI Model Repository.

Finally, for each I/O interface type, a dedicated mark-down document with the specification of the interface was created in the MARVEL GitLab Repository (Sections 3.4 and 3.6) and was regularly updated (Figure 50).



AV-Registry

A database that can store JSON objects, coupled with a REST API to allow MARVEL components to access information related to available AV sources in each MARVEL use case / deployment. The db.json file contains key-value pair info for each source which can be retrived with REST API calls.

Building the image

```
#clone the repository
git clone git@bitbucket.org:itmlgr/marvel-avregistry.git
cd marvel-avregistry
docker build -t avregistry:latest .
```

Spinoff a container

```
docker run -it -p 3000:3000 --name avregistrytest avregistry:latest
```

Retrieve all the available AV sources

```
root@dawn:~# curl localhost:3000/sources
```

```
[
    {
        "id": "Cam-GRN4-CCTV-17",
        "type": "Camera",
        "AVSourceType": "CCTV",
        "owner": "GRN4",
        "cameraNum": "03",
        "cameraName": "GRN4-CCTV-03",
```

Figure 50: A mark-down specification document for the AV Registry REST API

5.2.1 Onboard AV access

IFAG has developed an 8-microphone version of the Audiohub – Nano to meet the requirements needed by the MARVEL project partners. It is a dual-PCB stacked design. It consists of 8 IM69D130 MEMS microphones that are arranged in a circular pattern to allow for optimal direction finding.

It has the following features:

- Power source: USB connector
- 8x IM69D130 microphones
- Up to 48 kHz frequency sampling.
- Up to 24bit sampling resolution.
- Audio Data streaming via Wi-Fi
- Wi-Fi Module (Murata 1DX Wi-Fi-Module)
- PSoC 64 processing board
- SD card

Since the microphone board is connected via a header, it is very flexible, and different microphone configurations and geometries can be evaluated by developing additional boards. The PSoC 64 is the processing unit. It will be delivered with a default firmware streaming the raw 8-channel audio data via Wi-Fi to an external cloud. The signal acquired from the 8 microphones can be also stored in an external micro-SD card (i.e., there is no Wi-Fi connection available). The firmware capabilities of this board can be further programmed and debugged for extended functionality via JTAG or SWD. D4.4 presents a detailed description of the board.

5.2.2 AV Registry access

The AV Registry component exposes a REST API that is accessible by all services deployed in the MARVEL Kubernetes cluster. It allows them to retrieve information related to available AV sources in each MARVEL use case / deployment. The REST API foresees two calls that can be used to (1) retrieve all stored AV source documents as a JSON array and (2) retrieve a specific AV source document as a JSON object by referring to a particular AV source id.

Example for retrieving all AV source documents as a JSON array:

```
curl avregistrypod.svc:3000/sources/
```

Example for retrieving a specific AV source JSON object by providing the Camera object id:

```
curl avregistrypod.svc:3000/sources/"id"
```

The REST API returns the requested information in the form of JSON documents that follows the specification of the Camera data model (Section 5.3.2.1).

5.2.3 AV streaming

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. After analysing the available industry-standard protocols for live streaming of AV data, the MARVEL Consortium nominated the Real-Time Streaming Protocol (RTSP) as the protocol to be universally used for all needs of live streaming of AV data in 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 [46].

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 also 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.

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 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.

Specifically, the VideoAnony/AudioAnony component will read the visual/audio stream from either IP sensors with RTSP or directly from the live stream if the sensors are installed on the same device, perform the anonymisation algorithms and send the anonymised video/audio stream to a public server implementing the RTSP protocol [47].

In the case of VideoAnony reading from IP sensors, it reads the RTSP streaming data from an IP camera using OpenCV and then performs face and number plate detection and anonymisation using deep learning models. With OpenCV compiled with the GStreamer

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

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

support, VideoAnony then sends the anonymised visual stream to the server at “rtsp://ip:8554/anonystream”, where ip:8554 is the address of the TCP/RTSP listener. When the server receives an RTP packet, it extends the RTP packet header by adding a one-byte extension, as defined in Section 5.3.1 of [RFC3550] [48], with the absolute timestamp of when this packet was processed by the server. A similar procedure applies to AudioAnony too, while AudioAudio can directly access the microphone stream as it is installed on Edge devices in R2.

Clients that want to read the anonymised live visual stream must establish a connection with the server. The particular transport protocol (i.e., UDP, TCP, or UDP multicast) is chosen by the client while handshaking with the server. The transmission of the anonymised video between the clients and the server takes place via a push-based protocol. Once a client establishes a connection with the server, the RTSP server transmits packets to the client until the latter ends the session.

5.2.4 DatAna: AI inference result publication

As explained in section 4.3.1, the process of publishing the results of the AI inference models is done as messages in specific topics in MQTT brokers located typically in the same infrastructure where the models are running (edge, fog or cloud). The inference models subscribe to the AV streams and after the processing, they output the results in their respective topics in MQTT (named as per the acronym of the component in MARVEL, e.g., “SED” topic for the SED component). This way the inference models are decoupled from the rest of the components of the pipeline facilitating their integration. The output of each inference model can be different and communicated to the DatAna developers to allow further transformation, although some specific fields must be part of the output:

- “id”: Unique identifier of the entity. Formatted according to the NGSI standard. IDs for the specific event can be automatically assigned by component.
- “topic”: This field indicates the name of the queue the message is intended to.
- “owner”: A field containing a JSON encoded sequence of characters referencing the unique Ids of the owner(s). In the case of MARVEL, it is the use case name (e.g., MT1).
- “type”: The type of the entity. It must be one of the following values: "MediaEvent", “Alert” or “Anomaly”.
- “timestamp” or “startTime” / ”endTime”: Absolute timestamp for the event start/end (if the event is within a time span, or a single timestamp if it is a single time). Following the ISO8601 UTC format.
- “dateProcessed”: Timestamp of the processing of the entity by the inference model.
- “detectedBy”: Name of the MARVEL infrastructure node (edge, fog or cloud) that hosts the AI component that produces the detected event.
- “cameraId”: Unique identifier of the AV device (e.g., camera or microphone) from where the AV stream was taken as in the AV registry.
- “MLModelId”: The id of the MLModel that generated this event. It must be one of the models defined in the AV Registry.

[48] https://datatracker.ietf.org/doc/html/rfc3550#section-5.3.1

5.2.5 DatAna: AI result consumption

This I/O Interface type refers to cases where a MARVEL component needs to consume raw inference results from an AI component that resides on the same infrastructure node. This local interaction between the components is not direct but decoupled and it is enabled by the DatAna MQTT broker that is also hosted on the same infrastructure node. More specifically, the AI component publishes its inference results on a dedicated topic on the DatAna MQTT broker. The component that needs to consume the inference results subscribes to this topic of the DatAna MQTT broker and receives the results in real time as MQTT messages.

In R2, this is applicable to the interaction between (i) CATFlow and TAD, (ii) CATFlow and RBAD, (iii) AAC and GPURegex and (iv) YOLO-SED and Arduino Proxy (GRN1).

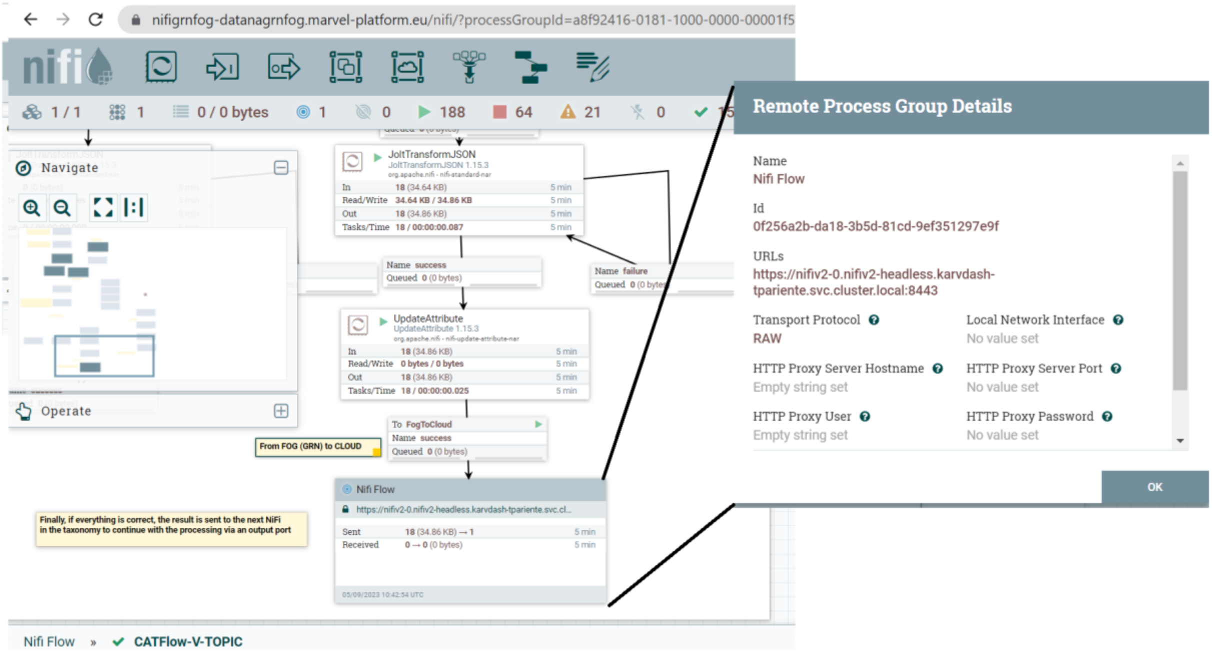
5.2.6 DatAna Inter-Agent Communication

Once the inference models provide their outputs to the nearest MQTT broker, DatAna provides specific data flows for each inference result to read these data and transform it to the required input by MARVEL in the form of the Smart Data Models for Alert, Anomaly or MediaEvent.

In order to do that, DatAna provides data flows deployed at the nearest NiFi instance to where the inference is running (edge, fog or cloud) that subscribe to each of the MQTT topics for each specific queue, retrieve the message in a streaming fashion and process them to make the necessary transformation and enrichment (e.g., adding timestamps, normalising some values and moving the inference results-specific fields to a common “data” field of the resulting output.

Once processed in the previous step, the final result of the DatAna processing in each layer (edge, fog or cloud) is a set of messages in the agreed Smart Data Model format for the Alert, Anomaly or MediaEvent data models. These messages are routed using the NiFi Site-to-Site protocol (S2S) to the upper NiFi in the topology (e.g., from to the DatAna at the Edge to the DatAna at the Fog and then to the DatAna at the cloud) as presented in Figure 51.

The data from these remote instances (typically from other fog servers or edge devices) are directed to the NiFi in the cloud via specific NiFi output ports, as shown in the last Nifi processor depicted in Figure 51. In this depiction, which is a partial screenshot taken from one of the data flows deployed at the NiFi instance at the fog server located at GRN, the last processor is a NiFi Remote Process Group. This processor acts as an output port pointing to the URL of the NiFi instance in the cloud, thus enabling communication with the cloud layer.



nifigrnfog-datanagrnfog.marvel-platform.eu/nifi/?processGroupId=a8f92416-0181-1000-0000-00001f5

Remote Process Group Details

Name
Nifi Flow

Id
0f256a2b-da18-3b5d-81cd-9ef351297e9f

URLs
https://nifiv2-0.nifiv2-headless.karvdash-tpariente.svc.cluster.local:8443

Transport Protocol	Local Network Interface
RAW	No value set
HTTP Proxy Server Hostname	**HTTP Proxy Server Port**
Empty string set	No value set
HTTP Proxy User	**HTTP Proxy Password**
Empty string set	No value set

OK

From FOG (GRN) to CLOUD

Finally, if everything is correct, the result is sent to the next NiFi in the taxonomy to continue with the processing via an output port

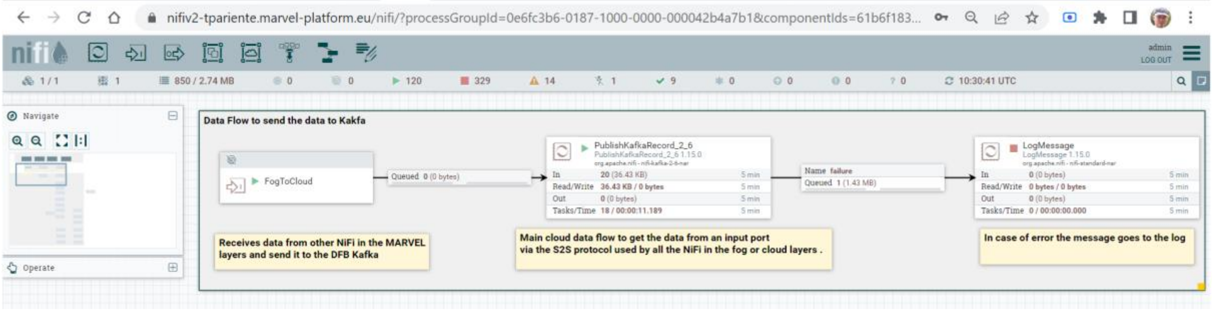
1 / 1
1
0 / 0 bytes
1
0
188
64
21
0
Navigate
Operate
JoltTransformJSON
JoltTransformJSON 1.15.3
org.apache.nifi - nifi-standard-nar
In 18 (34.64 KB) 5 min
Read/Write 34.64 KB / 34.86 KB 5 min
Out 18 (34.86 KB) 5 min
Tasks/Time 18 / 00:00:00.087 5 min
Name success
Queued 0 (0 bytes)
Name failure
Queued 0 (0 bytes)
UpdateAttribute
UpdateAttribute 1.15.3
org.apache.nifi - nifi-update-attribute-nar
In 18 (34.86 KB) 5 min
Read/Write 0 bytes / 0 bytes 5 min
Out 18 (34.86 KB) 5 min
Tasks/Time 18 / 00:00:00.025 5 min
To FogToCloud
Name success
Queued 0 (0 bytes)
Nifi Flow
https://nifiv2-0.nifiv2-headless.karvdash-tpariente.svc.cl...
Sent 18 (34.86 KB) → 1 5 min
Received 0 → 0 (0 bytes) 5 min
05/09/2023 10:42:54 UTC
Nifi Flow » CATFlow-V-TOPIC

Figure 51: DatAna – Communication between NiFi instances via output ports and S2S

The last step in the communication of messages is the reception of the messages at the cloud layer. The first processor depicted in Figure 51 (FogToCloud) is an input port located in the NiFi service deployed in the cloud, which allows receiving the incoming secured S2S communication between different NiFi instances using TLS. The messages do not suffer any alteration during the transfer between layers.

5.2.7 DatAna - DFB

Once in the cloud, DatAna forwards the messages to the DFB via the established Kafka topics of the same name as the field "topic" retrieved from the original output of the models. This is done via a specific NiFi processor that publishes to the specific topic of Kafka (PublsihKafkaRecord), as shown in the middle of Figure 52. Kafka is deployed as a service of the DFB in the Kubernetes cluster, ensuring the communication between the cloud NiFi and Kafka services.



nifiv2-tpariente.marvel-platform.eu/nifi/?processGroupId=0e6fc3b6-0187-1000-0000-000042b4a7b1&componentIds=61b6f183...

Data Flow to send the data to Kakfa

Receives data from other NiFi in the MARVEL layers and send it to the DFB Kafka

Main cloud data flow to get the data from an input port via the S2S protocol used by all the NiFi in the fog or cloud layers .

In case of error the message goes to the log

Figure 52: DatAna – From NiFi to the Kafka of the DFB

5.2.8 DFB - SmartViz

The DFB interacts with SmartViz in three different ways.

(a) **SmartViz real-time access to AI inference results aggregated at the DFB**. SmartViz needs to access the inference results that are being received by the DFB in real time in order to deliver them to the user. This is achieved through the subscription of SmartViz to the DFB Kafka topics, to which DatAna NiFi Cloud publishes all the collected AI inference results that have been transformed into the SDM-compliant MediaEvent, Anomaly and Alert data models.

(b) **SmartViz access to historical AI inference results aggregated at the DFB**. SmartViz needs to retrieve AI inference results that have been persistently stored at the DFB Elastic Search (ES) repository in order to deliver them to the user. For this purpose, a service has been developed for the DFB named "ES-Proxy" that exposes a REST API, which supports specific data queries to be made to the ES database. The supported queries return JSON documents and include the following indicative examples:

- Filter the AI inference results from a specific AI component in a specific time range.

```
curl -i -u -XPOST https://datafusion-es-proxy.marvel-
platform.eu/data/range/ -H "Content-Type: application/json; charset=utf-
8" -d '{ "index": "catflow-v", "start": "2021-11-26T15:32:00Z", "end":
"2021-11-26 15:32:45", "path": "vehicle.entry.ts" }'
```

- Filter the AI inference results from CATFlow in a specific time range and refer to a specific road lane to return information on the vehicles that were associated with the road lane.

```
curl -u -POST https://datafusion-es-proxy.marvel-platform.eu/data/lane/ -
H "Content-Type: application/json; charset=utf-8" -d '{ "start": "2021-
11-26 15:32:00", "end": "2021-11-26 15:32:45", "lane": "Mgarr - Triq iz-
Zebbiegh - Westwards" }'
```

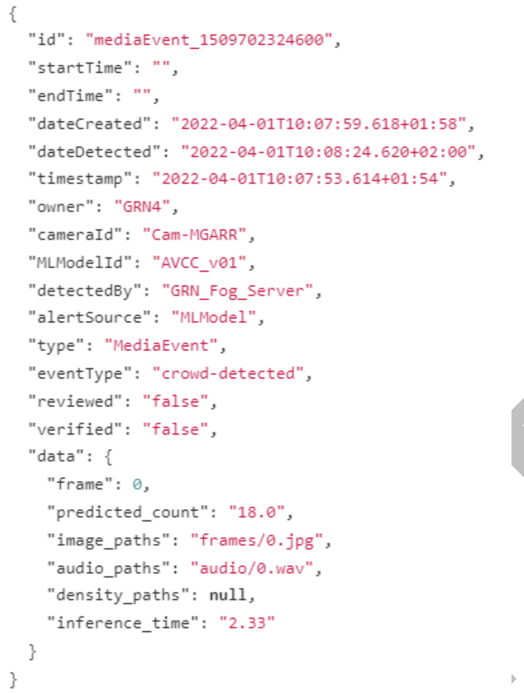
(c) **Update historical AI inference results at the DFB when verified by the SmartViz user**. SmartViz provides the option to the user to verify the accuracy of the AI inference results that are presented. This is particularly useful for collecting data labels for further AI training and optimisation of the MARVEL AI models. When a user of SmartViz verifies an AI inference result, SmartViz publishes a message to a dedicated topic ("InferenceVerification") on the DFB Kafka broker with the verification information that refers to a specific inference result id (Section 5.3.5). The DFB updates the corresponding inference result entry in the ES database according to the received information.

5.2.9 StreamHandler - DFB

StreamHandler subscribes to the DFB Kafka topics and consumes the AI inference results that are published by DatAna.

Based on information contained in each inference result (i.e., cameraId, startTime, endTime and timestamp if startTime, endTime are not present), StreamHandler constructs an AV file and stores it in the MinIO storage component. This AV file can be accessed by SmartViz upon request through the dedicated I/O Interface (Sections 5.2.10, 5.2.11) and by providing the corresponding inference result id.

An indicative message received from the DFB is presented in Figure 53.



```
{
  "id": "mediaEvent_1509702324600",
  "startTime": "",
  "endTime": "",
  "dateCreated": "2022-04-01T10:07:59.618+01:58",
  "dateDetected": "2022-04-01T10:08:24.620+02:00",
  "timestamp": "2022-04-01T10:07:53.614+01:54",
  "owner": "GRN4",
  "cameraId": "Cam-MGARR",
  "MLModelId": "AVCC_v01",
  "detectedBy": "GRN_Fog_Server",
  "alertSource": "MLModel",
  "type": "MediaEvent",
  "eventType": "crowd-detected",
  "reviewed": "false",
  "verified": "false",
  "data": {
    "frame": 0,
    "predicted_count": "18.0",
    "image_paths": "frames/0.jpg",
    "audio_paths": "audio/0.wav",
    "density_paths": null,
    "inference_time": "2.33"
  }
}
```

Figure 53: Indicative input data structure for StreamHandler's dfb-processor service

5.2.10 StreamHandler - SmartViz

The interaction between SmartViz and StreamHandler occurs through the REST API exposed by StreamHandler.

The URLs that SmartViz needs to use to access the REST APIs of the StreamHandler instances deployed at each pilot are presented in Figure 54.

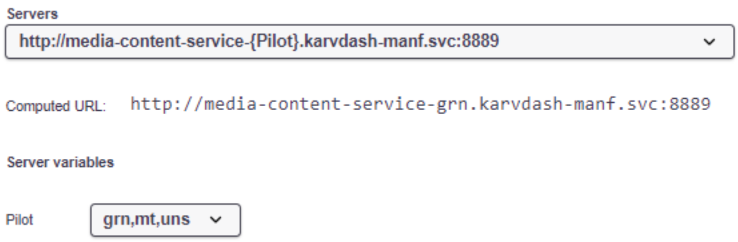
Figure 55 and Figure 56 demonstrate the two endpoints that SmartViz can use to interact with StreamHandler and request AV data segments that refer to a specific inference result that has been delivered to the user:

a) **On-demand video** request, where SmartViz provides the AV source ID and the start/end timestamps. In this case, the AV source ID refers to the AV source (Camera data model, Section 5.3.1), whose stream was analysed by the AI component to produce an inference result. The start/end timestamps refer to the absolute time period that the inference results correspond to.

b) **Event-based video** request, where SmartViz provides the event ID of an inference result, as published by DFB (described in subsection 5.2.9).

In the first case, StreamHandler produces an AV data segment that matches the requested time period by using the AV data that has already been stored in its storage component (MinIO) from the relevant segmented AV data files. StreamHandler stores the newly created AV data segment that it produces for the request also within the MinIO repository, then it returns a response to SmartViz that contains a URL, where the requested AV data segment can be accessed from.

In the second case, StreamHandler directly searches in the MinIO filesystem for a file with the provided event ID and then returns the URL of that file to SmartViz.



Servers

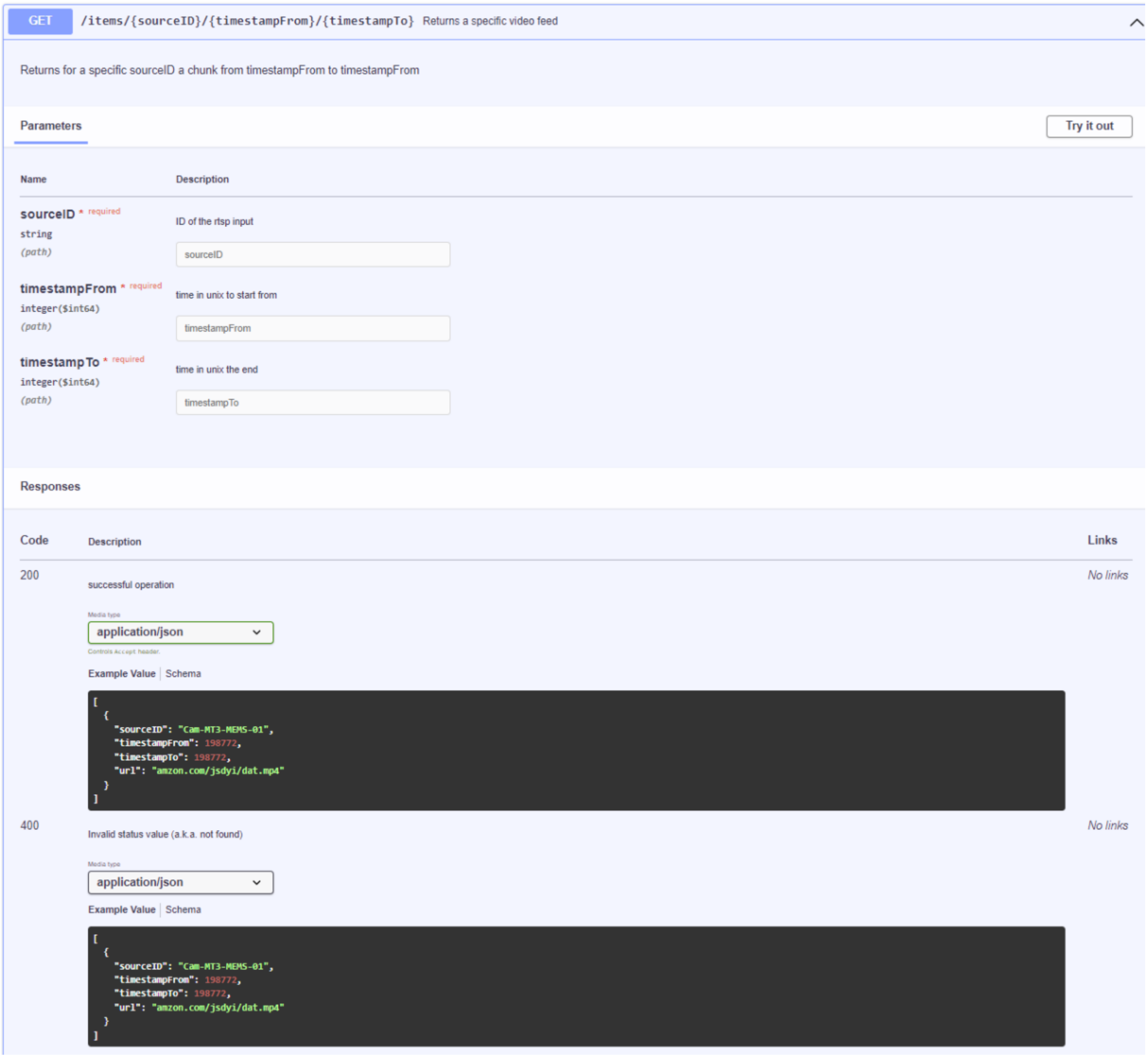
http://media-content-service-{Pilot}.karvdash-manf.svc:8889

Computed URL: `http://media-content-service-grn.karvdash-manf.svc:8889`

Server variables

Pilot: grn,mt,uns

Figure 54: StreamHandler URL for the API in each pilot



GET `/items/{sourceID}/{timestampFrom}/{timestampTo}` Returns a specific video feed

Returns for a specific sourceID a chunk from timestampFrom to timestampFrom

Parameters

Name	Description
sourceID * required string *(path)*	ID of the rtsp input sourceID
timestampFrom * required integer($int64) *(path)*	time in unix to start from timestampFrom
timestampTo * required integer($int64) *(path)*	time in unix the end timestampTo

Responses

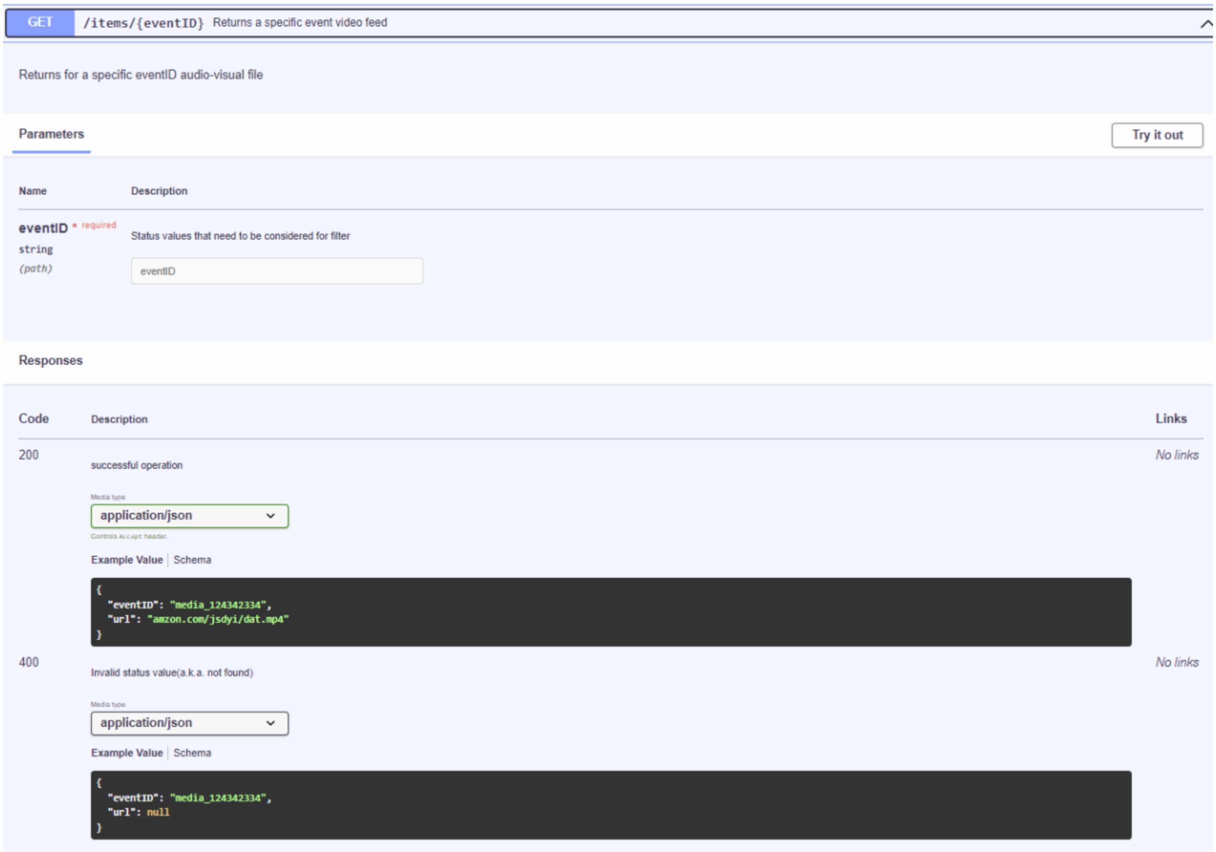
Code	Description	Links
200	successful operation Media type: application/json Controls Accept header. Example Value | Schema	*No links*

```
[
  {
    "sourceID": "Cam-MT3-MEMS-01",
    "timestampFrom": 198772,
    "timestampTo": 198772,
    "url": "amzon.com/jsdyi/dat.mp4"
  }
]
```

Code	Description	Links
400	Invalid status value (a.k.a. not found) Media type: application/json Example Value | Schema	*No links*

```
[
  {
    "sourceID": "Cam-MT3-MEMS-01",
    "timestampFrom": 198772,
    "timestampTo": 198772,
    "url": "amzon.com/jsdyi/dat.mp4"
  }
]
```

Figure 55: On-demand API endpoint example



/items/{eventID} Returns a specific event video feed

Returns for a specific eventID audio-visual file

Name	Description
eventID * required string (path)	Status values that need to be considered for filter eventID

Code	Description	Links
200	successful operation	No links
400	Invalid status value(a.k.a. not found)	No links

```
{
  "eventID": "media_124342334",
  "url": "amzon.com/jsdyi/dat.mp4"
}
```

```
{
  "eventID": "media_124342334",
  "url": null
}
```

Figure 56: Event-based API endpoint example

5.2.11 StreamHandler MinIO AV Data access

MinIO is used to store the AV files created by StreamHandler. The origin of the data stored in MinIO can be from:

- Processing and segmenting live AV data streams from the AV sources. The segmentation process is based on a stable time interval that is configurable (e.g., every 10, 20, 30 seconds). The created files are stored in the MinIO in the format **SourceID_start-timestamp_end_timestamp.fileType**, where filetype is mp4 for the videos and mp3 for the audio streams. The definition of the fields SourceID, start_timestamp and end_timestamp have been explained in subsection 5.2.10.
- Requests to REST API/DFB-processor services. In both cases, the relevant components access the AV files stored in the MinIO, perform the necessary changes (AV segments' combination) and store the newly created AV file back in the MinIO. The newly created files path is later shared with SmartViz for remote access.

5.2.12 DFB - HDD

HDD provides optimisation recommendations to DFB, as per the application settings. More than one producers in Apache Kafka can be supported., with one Kafka topic per each AI component. Also, more than one consumer can be supported. Consumers subscribe to the topics where inference data results are published. In order to handle those application settings as input, but also to provide back the optimised outputs, HDD interfaces with DFB and mutually exchange the data required for the operation in the JSON format. DFB is communicating to HDD the application setting inputs and, after the optimisation, HDD returns back to DFB the updated setting configuration as output. The HDD receives periodically the current Kafka topic partition configuration from the DFB and returns a recommendation for an optimised Kafka topic partition configuration. The running example of a file excerpt to be exchanged is the following:

```
[
    {
        "name": "topic1",
        "internal": false,
        "partitions": {
            "partition": 0,
            "leader": {
                "id": 2,
                "idString": "2",
                "host": "broker1.provider.com",
                "port": 9094,
                "rack": null
            },
            "replicas": {
                "id": 2,
                "idString": "2",
                "host": "broker1.provider.com",
                "port": 9094,
                "rack": null
            },
            "isr": {
                "id": 2,
                "idString": "2",
                "host": "broker1.provider.com",
                "port": 9094,
                "rack": null
            }
        }
    },
    {
        "name": "topic2",
        "internal": false,
        "partitions": {
            "partition": 0,
            "leader": {
                "id": 1,
                "idString": "1",
                "host": "broker2.provider.com",
                "port": 9094,
                "rack": null
            },
            "replicas": {
                "id": 1,
                "idString": "1",
                "host": "broker2.provider.com",
                "port": 9094,
                "rack": null
            },
            "isr": {
                "id": 1,
                "idString": "1",
                "host": "broker2.provider.com",
                "port": 9094,
                "rack": null
            }
        }
    }
]
```

Where name is the topic name, internal denotes whether the given topic is an internal topic or not, partitions are the running partitions for the given topic, partition is the partition number, leader is the leader partition, replicas are the replica partitions, id is the partition id, idString is the partition id in string format, isr are the in-sync replicas, host is the broker host, port is the host port, and rack is the host rack.

HDD exposes a REST API to facilitate the data exchange between DFB and HDD. The REST API supports:

- A **POST request** that can be used by the DFB to dispatch the current Kafka topic partition configuration document to HDD. Indicative example:

  ```
  curl -XPOST -d@partitions.json -H "cookie: $COOKIE" https://hddv1.svc/
  ```

  The command requires that the attachment `partitions.json` is available in the directory from which curl is called.

  After processing the input the service will send back a string saying "Input file saved"

- A **GET request** that can be used by the DFB to retrieve the optimised Kafka topic partition configuration document that is recommended by the HDD. Indicative example:

  ```
  curl -XGET -H https://hddv1.svc/ | python -m json.tool
  ```

  The expected output of this request is the same `partitions.json` file with modified content. Once this output is returned to the DFB, it is subsequently deleted from the HDD. If another GET request is issued, the service returns a "No output present" message.

Considering that it is not possible to update the DFB Kafka partitions during system operation since such an action could cause a disruption or failure of the DFB service, the following semi-automated pipeline is envisaged for updating the DFB Kafka topic partitions:

1. The DFB periodically retrieves the document that refers to the Kafka topic partition configuration that is currently applied.
2. The DFB issues a POST request to the HDD REST API to send the current partition configuration document to the HDD.
3. The DFB issues a GET request to the HDD REST API to receive the recommended partition configuration document from the HDD.
4. The DFB compares the original document to the one that has been recommended and returned by the HDD.
5. If the two partition configuration documents are found to be different, an email alert is generated by the DFB that is received by the DFB administrator.
6. The DFB administrator can proceed with manual actions to update the DFB Kafka partitions.

5.2.13 AI Model Repository access

The AI Model Repository (AIMR) is an internal service offering a common sharing point where all the models produced by the AI Training Pipeline. Specifically, all the AI models produced are stored on the AIMR such that they are accessible by the AI components running on the

MARVEL platform. The AIMR is based on MinIO and its internal organisation reflects the several use cases considered in MARVEL, i.e., each use case is associated with a bucket holding the model and a human-readable definition describing the main properties of the specific model. In this way, once a new or updated AI model is ready for a specific component, the component can download it from the AIMR. The AIMR serves also in the training and compression pipeline: when a compressed model is ready for deployment, it is uploaded on the AIMR and made available to the components that would need it.

It was deemed necessary that each AI model/weights saved on the AI Model Repository is coupled with metadata, packed into a JSON file that is called Model Descriptor. The Model Descriptor may contain information like the model's reference performance, its size, the minimum specs for running the model, etc. Its internal structure follows the MLModel data model (Section 5.3.4). These documents are human-readable, and their objective is to assist in selecting one of the AI models stored in the repository that fits for the specific purpose. The id referenced in an MLModel is also injected in all inference results that are produced through the use of the associated AI model.

Both the AI model's filename and the Model Descriptor (MLModel) have the same name, but the *.json* suffix needs to be attached to the latter.

Example

File containing the model: `avcc-headcount-au-full-v0.1`

File containing the model descriptor: `avcc-headcount-au-full-v0.1.json`

The interaction and management of the AIMR is done through the standard MinIO APIs. An example of how the AIMR interacts with some of the MARVEL components is provided in Section 5.1.12.

5.2.14 DataCorpus: AI training

The ingested datasets in the Data Corpus can be retrieved by the AI components that are deployed at the MARVEL Kubernetes cluster, facilitating their ML training and testing procedures. This interaction can involve two REST APIs to: i) retrieve a specific dataset based on the dataset ID, or ii) retrieve a specific snippet based on the snippet's ID.

In general, the following naming scheme is applied for the datasets' IDs.

DatasetID = <Project>.<DataProvider> [optional] _ <Device_id>.<Category (Video, Audio, or Video-Audio)>.<Annotated (Yes/No)>.<Anonymized (Yes/No)>.<Original data or Augmentation method><Date (dd-mm-yyyy)>.<Incrementing number starting from '1'>

Example ID: *MARVEL.GRN_Device1.Video.Y.Y.OriginalData.18-05-2022_1*

The datasets are composed of a list of snippets. Each snippet may have the core data file (video or audio), a file with annotation results, and/or a file with inference results. Similarly with datasets, the following naming scheme is used.

Therefore, the first API receives a dataset ID and returns a list with the underlying snippet IDs. If the dataset does not exist or if the dataset does not contain any snippets at the moment, an empty list is returned.

The second API receives a snippet ID and returns a list with the underlying file objects. Again, if the snippet does not exist or if the snippet does not contain any files at the moment, an empty list is returned.

SnippetID = DatasetID__<Incrementing number starting from '1'>

Example ID: *MARVEL.GRN_Device1.Video.Y.Y.OriginalData.18-05-2022_1__1*

Example of Snippet Files naming:

- Main data file:
    - *MARVEL.GRN_Device1.Video.Y.Y.OriginalData.18-05-2022_1__1.mpeg*
- Annotation file:
    - *MARVEL.GRN_Device1.Video.Y.Y.OriginalData.18-05-2022_1__1.txt*
- Inference resutlts file:
    - *MARVEL.GRN_Device1.Video.Y.Y.OriginalData.18-05-2022_1__1.score*

In the typical AI Training data flow, an AI component: i) asks for a dataset based on its ID, ii) retrieves the snippets list, and iii) iteratively retrieves the snippets' files.

5.2.15 FedL Server – FedL Client

The client and the server use gRPC protocol to communicate and exchange model parameters. The data model used is defined by the underlying flower [49] Federated Learning library. The data shared is a compressed numpy [50] array.

In addition to the model parameters, we use multiple metrics relevant to MARVEL-developed non-uniform sampling Federated Learning strategy designed with flaky communication in mind. Several model metrics (such as performance, number of relevant training data points, model gradient variance etc.) are provided by clients to the server so that the server can decide which updates to use. This also allows for varying asynchronous learning, since the frequency of the data can differ from client to client (e.g., real-time streaming data vs. batch processing of couple of days/weeks worth of data).

5.2.16 DFB - DataCorpus

The AI components can process data collected from the R2 pilot use cases at runtime with trained ML models and produce inference results. The user can also verify some of these results when they are delivered through SmartViz. In all cases, information is published on dedicated Kafka topics at the DFB. The scheme of these results complies with the data models in which

[49] https://flower.dev/

[50] https://numpy.org/

AI inference results are formatted by DatAna (MediaEvent, Alert, Anomaly), presented in Section 5.3.2.

At the Data Corpus end, a JAVA application is implemented to consume this information from the DFB Kafka and ingest it in its repository. Based on the *device/source ID* incorporated in the inference results (please refer to the Camera data model in Section 5.3.2.1) and the *date of the timestamps* in the MediaEvent, Alert, Anomaly data models, the Corpus can map the data to an existing AV dataset (i.e., we consider that there is a single dataset with the original recordings for each pilot device for each day), and based on the *start* and *end timestamps*, the Corpus can map the AV snippet file produced by StreamHandler to the relevant snippet entry within the dataset.

5.3 Data Models

The specification of I/O Interfaces and Data Models are the two main pillars in the integration of multiple components into a unified system to ensure its successful operation. This section presents the second pillar, i.e., the specification of Data Models.

The Data Models that have been specified and implemented for R2 ensure that there is a uniform and consistent way of structuring data that needs to be exchanged between MARVEL components or stored for subsequent access. The MARVEL R2 Data Models complement the I/O interfaces specified in Table 6 of Section 5.2. Table 6 also provides the associations between I/O Interfaces and Data Models.

In order to improve the visibility and acceptance of MARVEL results, we have made an attempt to align the data models we use for storage and handling of data to existing standardisation initiatives. One of the most promising initiatives is the Smart Data Models (SDM) program [51], started initially by FIWARE and now promoted by the FIWARE Foundation, TM Forum and IUDX, among others. This is one of the de-facto standards for data models in Europe and covers aspects related to smart applications, including Smart Cities, mobility and others relevant to MARVEL.

The SDM program[52] is a joint collaborative program to provide multisector, agile, standardised, free and open-licensed data models based on actual use cases and open standards, which is key for creating a global digital single market of interoperable and replicable (portable) smart solutions in multiple domains (e.g., smart cities, smart agrifood, smart utilities, smart industry). The SDM program is led by the FIWARE Foundation, TM Forum, IUDX, and is open to other entities to join. All data models are public and royalty-free nature of specifications. The SDM specification is hosted in GitHub [53] and contains JSON Schemas and documentation on Smart Data Models for different Smart Domains. For each Domain (industrial sector), there is a repository containing as submodules the link to the Subjects containing all the data models related. For each Vertical (Subject) there is a repository containing the data models related to that vertical.

The general principles of the SDM program are:

1. **Driven-by-implementation approach**: Specifications will be considered stable as soon as enough end-user organisations (e.g., cities) have validated them in practice.

[51] https://smartdatamodels.org/

[52] https://smartdatamodels.org/index.php/governance/

[53] https://github.com/smart-data-models/

2. **Open-closed**. Breaking changes to already approved specs is not allowed. Instead, new versions shall deprecate attributes, add new attributes, extend enumerations, etc.
3. **Open contribution**. Contributions open to anybody (not only members), while final decision-making corresponds to the administrators of the domains and Subjects. Management (currently TM Forum, FIWARE Foundation, and IUDX) could oppose some contributions if it does not meet coding guidelines.

The SDM program has been particularly active in the domain of Smart Cities, providing a series of data models that are applicable to IoT and AI technologies. In this context, during the early stages of integration and design activities of the MARVEL framework, it was decided that it would conform to and adopt the Smart Data Models (SDM) standard for its data modelling needs, wherever possible, as this approach was found to provide a strategic advantage to MARVEL.

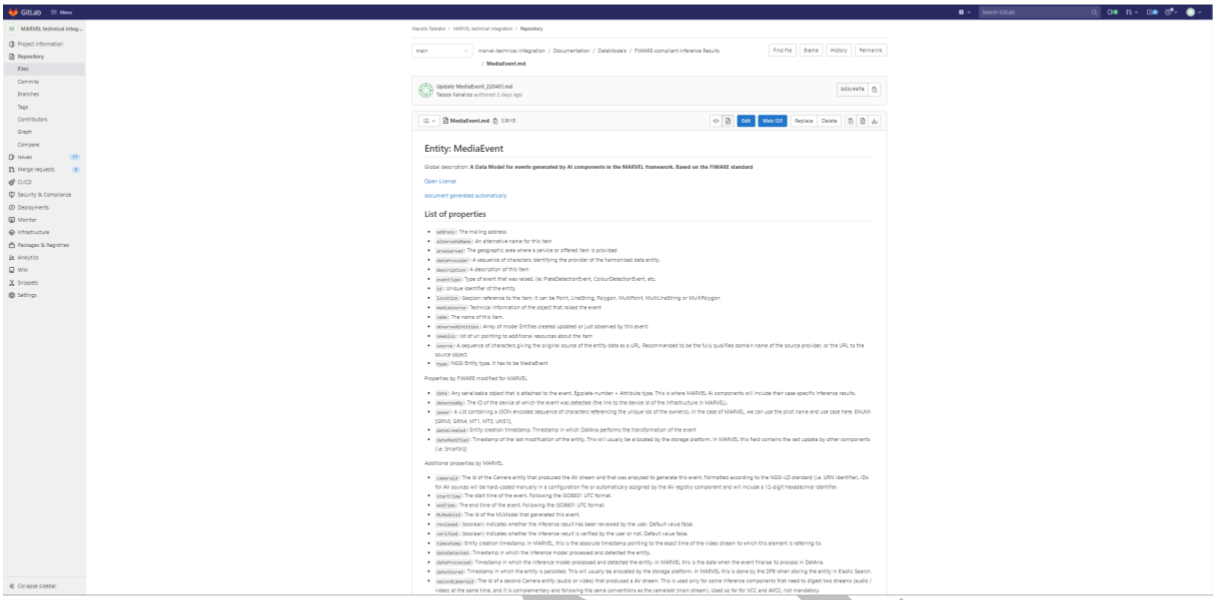
To that extent, the data models provided by SDM and in particular the ones within the SDM Smart City Domain were carefully examined to identify the ones that could most closely match the specific needs of MARVEL. The following SDM data models were promoted for use in the MARVEL framework:

- **Camera** (Section 5.3.1) This data model was selected for describing all MARVEL AV sources.
- **MediaEvent** (Section 5.3.2). This data model was selected for describing all MARVEL general-purpose AI inference results produced by AI components.
- **Alert** (Section 5.3.2). This data model was selected for describing all MARVEL AI inference results produced by the MARVEL AI components that assume the form of an alert for MARVEL users.
- **Anomaly** (Section 5.3.2). This data model was selected for describing all MARVEL AI inference results produced by the MARVEL AI components that assume the form of a detected anomaly.
- **MLModel** (Section 5.3.4). This data model was selected for describing all MARVEL AI models.

These SDM data models were found to be the most relevant for the needs of the MARVEL project, but the original specifications did not fully address all needs being considered. Therefore, it was found necessary to extend the specifications of these data models by re-purposing existing and adding new fields that could serve all remaining requirements. This was carefully performed so as not to break the compliance with the SDM standard.

The specifications of all SDM data models are structured as JSON documents and include a set of fields, which are categorised as optional and mandatory. The adaptations of the selected SDM data models that were performed for MARVEL preserved the original data model names and all optional and mandatory fields. Wherever the original definition of SDM data model fields was not sufficient to address all MARVEL requirements, an attempt was made to re-purpose existing fields by specifying the exact context in which they should be used within MARVEL. Such cases were explicitly categorised in the MARVEL data model specifications as “Properties that are modified for MARVEL”. In cases, where this was not possible, the new fields that were added for the needs of MARVEL were also categorised as optional (for use in MARVEL) and mandatory (for use in MARVEL). This process ensured that SDM-compliance is maintained and that all the essential information that is required for the successful operation of all MARVEL components is included in the MARVEL data models. In addition, all SDM conventions were implemented in the definition of new fields and in the specification of the field value types, syntax and format.

Finally, for each MARVEL data model, a dedicated mark-down document with the specification of the data model was created maintained in the MARVEL GitLab Repository (Sections 3.4 and 3.6, Figure 57).



Entity: MediaEvent

Global description: **A Data Model for events generated by AI components in the MARVEL framework. Based on the FIWARE standard**

Open License

document generated automatically

List of properties

- `address`: The mailing address
- `alternateName`: An alternative name for this item
- `areaServed`: The geographic area where a service or offered item is provided
- `dataProvider`: A sequence of characters identifying the provider of the harmonised data entity.
- `description`: A description of this item
- `eventType`: Type of event that was raised. (ie: PlateDetectionEvent, ColourDetectionEvent, etc.
- `id`: Unique identifier of the entity
- `location`: Geojson reference to the item. It can be Point, LineString, Polygon, MultiPoint, MultiLineString or MultiPolygon
- `mediaSource`: Technical information of the object that raised the event
- `name`: The name of this item.
- `observedEntities`: Array of model Entities created updated or just observed by this event.
- `seeAlso`: list of uri pointing to additional resources about the item
- `source`: A sequence of characters giving the original source of the entity data as a URL. Recommended to be the fully qualified domain name of the source provider, or the URL to the source object.
- `type`: NGSI Entity type. It has to be MediaEvent

Properties by FIWARE modified for MARVEL

- `data`: Any serializable object that is attached to the event. Eg:plate-number + Attribute type. This is where MARVEL AI components will include their case-specific inference results.
- `detectedBy`: The ID of the device at which the event was detected (the link to the device id of the infrastructure in MARVEL).
- `owner`: A List containing a JSON encoded sequence of characters referencing the unique ids of the owner(s). In the case of MARVEL, we can use the pilot name and use case here. ENUM: [GRN3, GRN4, MT1, MT3, UNS1].
- `dateCreated`: Entity creation timestamp. Timestamp in which DatAna performs the transformation of the event
- `dateModified`: Timestamp of the last modification of the entity. This will usually be allocated by the storage platform. In MARVEL this field contains the last update by other components (i.e. SmartViz)

Additional properties by MARVEL

- `cameraId`: The id of the Camera entity that produced the AV stream and that was analysed to generate this event. Formatted according to the NGSI-LD standard (i.e. URN identifier). IDs for AV sources will be hard-coded manually in a configuration file or automatically assigned by the AV registry component and will include a 12-digit hexadecimal identifier.
- `startTime`: The start time of the event. Following the ISO8601 UTC format.
- `endTime`: The end time of the event. Following the ISO8601 UTC format.
- `MLModelId`: The id of the MLModel that generated this event.
- `reviewed`: (boolean) Indicates whether the inference result has been reviewed by the user. Default value false.
- `verified`: (boolean) Indicates whether the inference result is verified by the user or not. Default value false.
- `timestamp`: Entity creation timestamp. In MARVEL, this is the absolute timestamp pointing to the exact time of the video stream to which this element is referring to.
- `dateDetected`: Timestamp in which the inference model processed and detected the entity.
- `dateProcessed`: Timestamp in which the inference model processed and detected the entity. In MARVEL this is the date when the event finalise its process in DatAna.
- `dateStored`: Timestamp in which the entity is persisted. This will usually be allocated by the storage platform. In MARVEL this is done by the DFB when storing the entity in Elastic Search.
- `secondCameraId`: The id of a second Camera entity (audio or video) that produced a AV stream. This is used only for some inference components that need to digest two streams (audio / video) at the same time, and it is complementary and following the same conventions as the cameraId (main stream). Used so far for VCC and AVCC, not mandatory.

Figure 57: A mark-down specification document for the MediaEvent data model

The following sections 5.3.1– 5.3.5 describe the implemented MARVEL data models in more detail. A complete documentation of the MARVEL data model specifications can be found in Appendix B.

5.3.1 AV Source Data Model (Camera)

The MARVEL *Camera* data model is a specialisation of the original SDM *Camera* data model [54] that was implemented to describe all AV data sources in MARVEL, including cameras, microphones and anonymisation components (VideoAnony, AudioAnony). Although microphones and anonymisation components do not match the context of the original SDM data model, the *Camera* data model was the only SDM data model that was relevant enough to describe all MARVEL AV sources. It was decided not to propose the generation of other, completely new, dedicated data models, but rather try to extend the Camera data model definition to account for such cases in order to preserve SDM compliance.

This data model was used for the definition of documents that describe the AV sources in each pilot (GRN, MT, UNS), which are managed by the AV Registry component (Section 4.1.3).

The MARVEL *Camera* data model includes fields that refer to the id, the type, the georeferenced position, the live RTSP stream URL, the video resolution, the video framerate, the audio channels and the type of encoding of the AV source among others.

The most notable fields, which are essential to the operation of the AI components are the following:

[54] https://github.com/smart-data-models/dataModel.Device/tree/master/Camera

- **id**: This is a unique id that is assigned to each MARVEL AV source. AI components are required to reference this id in each inference result they produce so that it is possible to subsequently associate each inference result to the AV source, whose streaming data were used by the AI component to generate the specific inference result.
- **streamURL**: This is the complete network path to the live RTSP-based AV stream of the corresponding AV source, which is required by AI components, StreamHandler and SmartViz to access the necessary AV data.

A complete documentation of the specification of the MARVEL Camera data model is provided in Appendix B.

5.3.2 SDM-compliant AI Inference Result Data Models (MediaEvent, Alert, Anomaly)

A crucial aspect of the 'AI Inference Pipeline' is the production, transmission, storage and consumption of AI inference results. Therefore, it was of paramount importance to specify data models for efficiently structuring these results to maintain interoperability across the MARVEL components that manage such data. Three data models are specified by MARVEL for this purpose, namely the *MediaEvent*, *Alert* and *Anomaly* data models, which are specialisations of their original SDM data model counterparts with the same names.

All raw AI inference results produced by the MARVEL AI components in the 'AI Inference Pipeline' are collected by DatAna. DatAna is responsible for transforming these raw inference results into the three SDM-compliant aforementioned data models (MediaEvent, Alert, Anomaly) before relaying them to the DFB for persistent storage and consumption by other components (e.g., SmartViz, StreamHandler, Data Corpus).

While the three data models have a different focus, all of them share a common set of fields. The most notable ones are:

- **id**: A unique id assigned to each inference result by the AI component that produced the original raw inference result counterpart.
- **cameraId**: The id of the Camera entity (AV source) that produced the AV stream and that was analysed to generate the inference result.
- **owner**: In the case of MARVEL, the pilot and use case code is used. ENUM: [GRN3, GRN4, MT1, MT3, UNS1].
- **detectedBy**: The ID of the device at which the event was detected. In the case of MARVEL, a unique id of the infrastructure node that hosts the AI component instance that produced the original raw inference result counterpart is used.
- **data**: In the case of MARVEL, this field is used to encapsulate any case-specific data structures, produced by each MARVEL AI component. This field can possibly include multiple fields and nested structures.
- **startTime**: Within MARVEL, in case the inference result refers to a time period, this is the absolute timestamp pointing to the start time of the period. Following the ISO8601 UTC format.
- **endTime**: Within MARVEL, in case the inference result refers to a time period, this is the absolute timestamp pointing to the end time of the period. Following the ISO8601 UTC format.
- **timestamp**: Within MARVEL, in case the inference result refers to a time instant, this is the absolute timestamp pointing to the exact time this instant. Following the ISO8601 UTC format.
- **MLModelId**: The id of the MLModel that was applied by the AI component that produced the original raw inference result counterpart.

In addition, the three data models used for SDM-compliant AI inference results in MARVEL, include certain time/date fields that are used for logging the time of sequential intermediate steps in the 'AI Inference Pipeline', which are also foreseen to be used in the context of the MARVEL benchmarking activities (T5.4). **Table 7** presents these fields and indicates whether these fields are foreseen by the original SDM data model specification or have been introduced in the MARVEL specification. The values of all these fields follow the ISO8601 UTC format[55], with the following conventions:

- Syntax: YYYY-MM-DD´T´HH:MM:SS.SSS´Z´
- Example: "2022-04-01T10:08:24.620+02:00"
- The last section (+02:00) denotes the offset from the UTC time zone.
- In case the timestamp is in the UTC time zone, the following syntax is used: "2022-04-01T10:08:24.620**Z**".

Table 7: Time/Date fields used in the MARVEL Inference Result Data Models and presence in original SDM data models

Time field	**dateDetected** Date of detection of the event in the inference model		**dateCreated** Date of entry to DatAna		**dateProcessed** Date of exit from DatAna to the DFB		**dateStored** Date of storage in Elastic Search		**dateModified** Last update by other components (i.e., SmartViz)	
AI Inference Result type	SDM	MARVEL	SDM	MARVEL	SDM	MARVEL	SDM	MARVEL	SDM	MARVEL
Alert		X	X	X		X		X	X	X
Anomaly	X	X	X	X		X		X	X	X
MediaEvent		X	X	X		X		X	X	X

Documents belonging in these three data models can be distinguished by the value of the field "type", which receives the value "MediaEvent", "Alert" or "Anomaly" in each case.

The following sub-sections present more details regarding the MARVEL ***MediaEvent***, ***Alert*** and ***Anomaly*** data models.

5.3.2.1 MediaEvent Data Model

The MARVEL ***MediaEvent*** data model is a specialisation of the original SDM *MediaEvent* data model[56]. The purpose of the model is to specify any potential event based on the results of the 'AI Inference Pipeline'. The MediaEvent data model is used for events or specific situations detected by the inference models that cannot be categorised as alerts or anomalies (e.g., crowd counting at a given moment of time). Events can be of different "eventType" or category (e.g., "crowd-detected"), which basically covers all the spectrum of messages issued by the AI models not falling under the Anomaly or Alert data models.

The mandatory properties are: eventType, id, type, cameraId, startTime, endTime, timestamp (either timestamp or startTime/endTime are mandatory), owner, data, MLModelId, detectedBy, dateDetected, dateCreated, reviewed, verified.

The complete documentation of the MediaEvent data model specification can be found in Appendix B.

[55] https://www.iso.org/iso-8601-date-and-time-format.html

[56] https://github.com/smart-data-models/dataModel.Multimedia/blob/master/MediaEvent/doc/spec.md

5.3.2.2 Alert Data Model

The MARVEL ***Alert*** data model is a specialisation of the original SDM *Alert* [57] data model. The purpose of the model is to indicate potential alerts based on the results of the inference pipeline, which need to be delivered to the user.

The mandatory properties are: alertSource, category, id, type, cameraId, startTime, endTime, timestamp (either timestamp or startTime/endTime are mandatory), owner, data, MLModelId, detectedBy, dateDetected, dateCreated, reviewed, verified.

The complete documentation of the Alert data model specification can be found in Appendix B.

5.3.2.3 Anomaly Data Model

The MARVEL ***Anomaly*** data model is a specialisation of the original SDM *Anomaly* [58] data model. The purpose of the model is the indication of potential detected anomalies based on the results of the inference pipeline, i.e., rare events that significantly differ from the events that are normally expected (e.g., a loud sound detected by a microphone in a quiet area).

The mandatory properties are: anomalousProperty, id, type, cameraId, startTime, endTime, timestamp (either timestamp or startTime/endTime are mandatory), owner, data, MLModelId, detectedBy, dateDetected, dateCreated, reviewed, verified.

The complete documentation of the Alert data model specification can be found in Appendix B.

5.3.3 Raw AI Inference Result Data Models

The MARVEL AI components implement different data models for structuring their raw output of inference results according to the nature and objectives of the respective AI model. However, the raw AI inference result data models are all required to adhere to certain principles in order to allow the transformation of the raw results into the SDM-compliant equivalents by DatAna. Therefore, the use of certain fields such as **id, cameraId, owner, detectedBy, timestamp, startTime, endTime, dateDetected, MLModelId** that are derived from the SDM-compliant data models are mandatory. Other fields that are case-specific for each AI component are transferred to the internal structure of the “data” field when the raw result is transformed into the SDM-compliant equivalent.

The following sub-sections present the data model for the raw inference results produced by each MARVEL AI component.

5.3.3.1 CATFLow output Data Model

The GRN CATFlow component produces an output message whenever a vehicle or pedestrian is detected. The vehicle and pedestrian detections are separated into two types of data models.

The **CATFlow Vehicles data model** provides information on the detected vehicles (e.g., entry/exit points, trajectory in the pixel space of the analysed video frames, road lane, speed, etc).

Main fields: location, vehicle.type, vehicle.name, vehicle.entry, vehicle.lane_flow_uuid, vehicle.distance, vehicle.speed_kmh, vehicle.trajectory_points.

[57] https://github.com/smart-data-models/dataModel.Alert/blob/master/Alert/doc/spec.md

[58] https://github.com/smart-data-models/dataModel.Alert/blob/master/Anomaly/doc/spec.md

The **CATFlow Pedestrian data model** provides information on the detected pedestrians (e.g., trajectory in the pixel space of the analysed video frames, start/end time of detection, etc)

Main fields: location, pedestrian.start_ts, pedestrian.end_ts, pedestrian.time_seconds pedestrian.trajectory_points

A complete documentation of the specification of the CATFlow Vehicles and CATFlow Pedestrians data models can be found in Appendix B.

5.3.3.2 TAD output Data Model

The GRN TAD component produces an output message whenever an anomalous speed is detected from the analysis of the raw inference results produced by CATFlow.

The **TAD data model** provides information on the detected anomaly.

Main fields: category, thresholdBreach

The CATFlowid fields and the trajectory points fields were added to the TAD data model in order to allow a reference to the original CATflow trajectory and output the trajectory points of anomalous tracks.

A complete documentation of the specification of the TAD data model can be found in Appendix B.

5.3.3.3 ViAD / AVAD output Data Model

The role of the ViAD and AVAD components by AU is to detect anomalies by analysing video frames either exclusively or in combination with audio analysis respectively. For each analysed AV segment, they produce a message that refers to the time period corresponding to that video frame. According to whether an anomaly is detected or not, their raw inference results are transformed to a document following the MediaEvent or Anomaly data model respectively.

The **ViAD / AVAD data model** provides information on whether an anomaly has been detected by assigning a corresponding value to a field named "predictedAnomaly" which is of boolean type.

A complete documentation of the specification of the ViAD / AVAD data model can be found in Appendix B.

5.3.3.4 VCC / AVCC output Data Model

The role of the VCC and AVCC components by AU is to analyse video frames either exclusively or in combination with audio analysis respectively to provide the predicted number of persons in the input AV segment as well as the probability of human presence in the areas of the analysed video segment.

The probability of human presence is delivered as normalised values that correspond to the pixels of the analysed video frame in the form of a heatmap (thermographic image) that can be used for visualisation purposes.

For each analysed AV segment, VCC and AVCC produce an output message that refers to the time period corresponding to that video segment and contains the predicted number of persons in the input image. The heatmap is calculated and included in the output message periodically and not continuously.

The **VCC / AVCC data model** provides information on the number of detected people and a heatmap. In R2, the heatmap is encoded as an ASCII character string using the Base64 scheme.

Main fields: predictedCount, heatmap

A complete documentation of the specification of the VCC / AVCC data model can be found in Appendix B.

5.3.3.5 SED output Data Model

The raw inference results of the MARVEL SED component follow the **SED output data model**, which specifies how the detected sound events are represented in the pipeline.

Each event has absolute timestamp fields “startTime” and “endTime” in ISO8601 UTC format. The “label” field stores the event class label in text format.

A complete documentation of the specification of the SED data model can be found in Appendix B.

5.3.3.6 AT output Data Model

The raw inference results of the MARVEL AT component follow the **AT output data model**, which specifies how the audio tags for segments of an audio signal are represented in the pipeline.

Each tag has absolute timestamp fields “startTime” and “endTime” in ISO8601 UTC format. The “label” field stores the tag label in text format. The model is similar to the MARVEL SED output data model, however, “startTime” and “endTime” are now on a fixed time grid defined by the inference model and there can be multiple tags active during the same time interval.

A complete documentation of the specification of the AT data model can be found in Appendix B.

5.3.3.7 AAC output Data Model

The raw inference results of the MARVEL AAC component follow the **AAC output data model**, which specifies how the audio captions extracted from an audio signal are represented in the pipeline.

Each caption has absolute timestamp fields “startTime” and “endTime” in ISO8601 UTC format. The “label” field stores the audio caption in text format.

A complete documentation of the specification of the AT data model can be found in Appendix B.

5.3.3.8 SELD output Data Model

The raw inference results of the MARVEL SELD component follow the **SELD output data model**, which specifies how detected localised sound events are represented in the pipeline.

Each event has absolute timestamp fields “startTime” and “endTime” in ISO8601 UTC format. The “label” field stores the event class label in text format.

The SELD data model also uses the following fields:

- “azimuthAngle”, azimuth of the sound source in relation to the mic array orientation (-180 to +180 degrees)
- “elevationAngle”, elevation of the sound source in relation to the mic array orientation (-90 to +90 degrees)
- “azimuthVector”, vector showing direction of sound source in GPS coordinates, a list of two arrays with two floats [[lat1, lon1], [lat2,lon2]].

A complete documentation of the specification of the AT data model can be found in Appendix B.

5.3.3.9 YOLO-SED output Data Model

The role of the YOLO-SED component is to detect anomalies related to vulnerable road users based on simulateneous processing of audio and video information that are based on the SED audio analysis and the YOLO object detector respectively.

The **YOLO-SED data model** provides information on whether a vulnerable road user has been detected.

The YOLO-SED inference results are meant to alert the competent authorities and therefore they are subsequently transformed by DatAna to events that follow the SDM-compliant “Alert” data model.

Each event has absolute timestamp fields “startTime” and “endTime” in ISO8601 UTC format.

Additionally, the “category” and “subcategory” fields are used to provide information related to the classification of the detected event. These fields are derived from the original SDM “Alert” data model. Typical value for “category” is “traffic” and indicative values for “subcategory” are “injuredBiker”, “pedestrianOnRoad”, “bikerOnRoad”.

A complete documentation of the specification of the YOLO-SED data model can be found in Appendix B.

5.3.3.10 RBAD output Data Model

The role of the RBAD component is to analyse inference results generated by the CATFlow component and detect anomalies based on pre-specified rules.

The **RBAD data model** provides information on whether an anomaly has been detected by assigning a corresponding value to a field named “predictedAnomaly” which is of boolean type.

Each event has absolute timestamp fields “startTime” and “endTime” in ISO8601 UTC format.

Additionally, the field “anomalousProperty” contains information related to the type of detected anomaly, e.g., 'bus_not_on_schedule', 'bicycle_not_on_path', 'large_veh_rush_hour', 'jaywalking', etc.

A complete documentation of the specification of the YOLO-SED data model can be found in Appendix B.

5.3.3.11 VAD output Data Model

The raw inference results of the MARVEL VAD component follow the **VAD output data model**, which represents the timestamps of the detected audio event segments. These events can be either Speech (voice activity) or Music. Therefore, each output contains absolute timestamp fields “**startTime”** and “**endTime”** in ISO8601 UTC format, along with the field “**category**” storing the event category label in textual format.

A complete documentation of the specification of the VAD data model can be found in Appendix B.

5.3.4 MLModel Data Model

The SDM MLModel data model is adopted for MARVEL with few modifications. The purpose of the MLModel data model is to describe elements of a machine learning model in the form of metadata. The documents that are based on this data model are used to complement the AI models stored in the MARVEL AI Model Repository as Model Descriptors, following the same naming convention that is introduced for the AI models (Section 5.2.13).

MARVEL introduces a new field (“**data**”) to the SDM data model that refers to the payload containing extra ad-hoc complementary metadata needed by the MARVEL components, which are not covered in the other SDM existing fields. The “data” field encapsulates other fields such as file_name_weights, model_definition, code, val_accuracy, loss, training_epochs.

The MLModel properties that are considered mandatory for MARVEL are id, type, acceptableDataSources, algorithm, dateCreated, dateModified, description, id, inputAttributes, mlFramework, name, outputAttributes, outputDataTypes, source, typeOfAlgorithm, version, data, data. file_name_weights.

A complete documentation of the specification of the MLModel data model can be found in Appendix B.

5.3.5 Inference Verification Message Data Model

According to the R2 design, AI inference results that are persistently stored at the DFB ES need to be updated when they are verified by the user of SmartViz (case c in Section 5.2.8). More specifically, when a SmartViz user reviews a specific inference result in SmartViz and provides feedback, i.e., determines if the result is verified or not, SmartViz publishes a message to a Kafka topic in the DFB ("InferenceVerification"). The DFB obtains these messages and updates the respective inference result entries stored in the DFB ES accordingly. The DataCorpus also subscribes to this topic to obtain these messages and store them so that they can be used for labelling in the context of future AI training purposes.

In this context, a data model was prepared for the inference verification messages published by SmartViz. The data model follows the JSON syntax and contains the following fields:

- `index`: The index of ES, under which the inference result has been stored (a separate index is available for each MARVEL AI component).
- `inferenceResultId`: The id of the inference result about which the user has provided verification. The inference result type can be "MediaEvent", "Alert" or "Anomaly".
- `reviewed`: (boolean) Indicates whether the inference result has been reviewed by the user.
- `verified`: (boolean) Indicates whether the inference result is verified by the user or not.
- `verificationDate`: Date and time of verification of the inference result in ISO8601 UTC format.

The fields “reviewed” and “verified” are already present in the inference results stored at the DFB ES and their values are updated according to the information in the messages published by SmartViz at the “InferenceVerification” topic, following the aforementioned data mode.

A complete documentation of the specification of the Inference Verification Message data model can be found in Appendix B.

5.4 R2 Architecture instantiation per use case

5.4.1 GRN1 – Safer roads: AI Inference runtime and deployment view

Figure 58 provides the deployment and runtime view of the MARVEL architecture for the GRN1 use case.

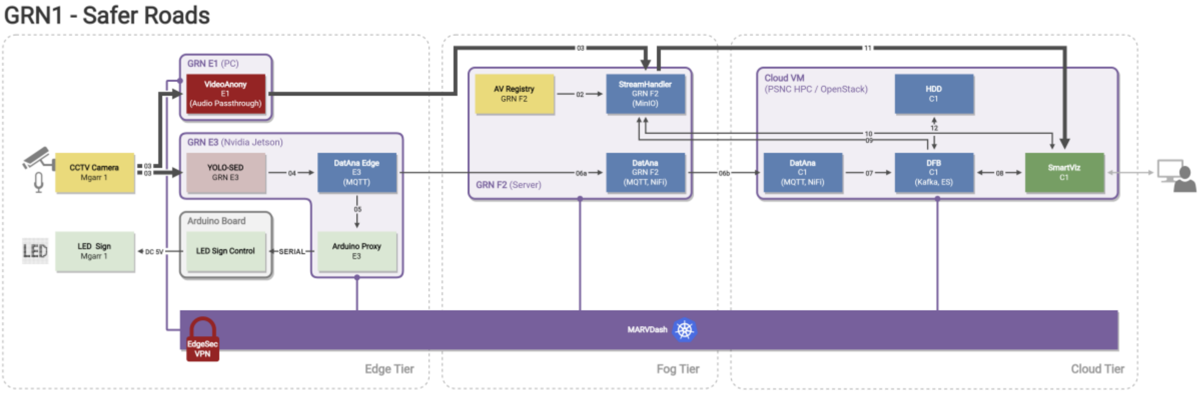


Figure 58: MARVEL R2 deployment and runtime view of the MARVEL architecture for GRN1 – Safer roads *(annotation label descriptions in Table 6, Section 5.2)*

This section describes the deployment, operation and interactions between components that are implemented for GRN2.

Two software components have been exclusively developed for the specific needs of GRN1:

(a) Arduino Proxy. Relays messages from an MQTT broker to an Arduino board via serial protocol communication.

(b) **Arduino LED control**. Arduino script that controls an LED board based on incoming messages.

These components facilitate the control of the **GRN1 LED board** based on the inference results produced by the **YOLO-SED** component.

In **GRN1**, an audio and video stream are obtained from a **CCTV camera** at the Mgarr location in Malta.

A **YOLO-SED** instance hosted on a Jetson device at the Edge (**GRN E3)** at the Mgarr location receives the AV stream from the **CCTV camera** at Mgarr over RTSP. **YOLO-SED** analyses the stream's audio and video section to produce inference results that correspond to detection of vulnerable road users.

The **YOLO-SED** instance publishes these inference results in real time to a dedicated topic of a **DatAna MQTT broker** residing at the same host Jetson device as the **YOLO-SED** instance (**GRN E3**).

An **Arduino Proxy** instance is also hosted on the Jetson device (**GRN E3**) and implements an MQTT client to consume the inference results produced by **YOLO-SED** and published on the **DatAna MQTT broker**. The Arduino Proxy converts the messages to suitable equivalent signals for transmission over a Serial protocol and communicates them to an Arduino board.

The **Arduino LED control** script is hosted on an Arduino board and receives the signals from the **Arduino Proxy** component over a Serial protocol communication channel. Based on the information it receives, it issues commands to the **LED board** it controls to display appropriate text messages that can be seen by road users. The commands are transmitted via 5V DC current through the Arduino board pins.

A **VideoAnony** instance hosted on a PC at the Edge (**GRN E1)** at the Mgarr location receives the AV stream from the camera at Mgarr.

During system initialisation, the component instances that need to consume the produced anonymised AV data streams (i.e., **StreamHandler**) request the metadata details of the **VideoAnony** AV source from the **AV Registry** (**GRN F2**) via corresponding REST API calls.

DatAna NiFi nodes collect the inference results from the corresponding **DatAna MQTT brokers** by subscribing to the respective MQTT topics. Specifically, the **DatAna NiFi Edge node** on **GRN F2** receives the inference results that are published on the **DatAna MQTT Edge broker** on **GRN E3**. Each **DatAna NiFi node** performs certain operations on the collected raw inference results from all AI components to transform them to SDM-compliant data models (for more details, see Section 5.3.2). Then, each **DatAna NiFi node** pushes the transformed inference results to the **DatAna NiFi node** on the higher E/F/C layer, i.e., (i) the **DatAna NiFi Edge nodes** push the results to the respective **DatAna NiFi Fog node** and (ii) the **DatAna NiFi Fog nodes** push the results to the **DatAna NiFi Cloud node**.

The **DatAna NiFi Cloud node** publishes the transformed, SDM-compliant inference results it collects from all layers to dedicated Kafka topics at the **DFB** (a separate topic is used for the inference results of each AI component that produced them), which is deployed at the **PSNC HPC**. The **DFB** persistently stores the received results on the **DFB ES** repository.

SmartViz is deployed at the **PSNC HPC** and accesses the incoming inference results at the **DFB** in real time by subscribing to the Kafka topics where they are published by the **DatAna NiFi Cloud node**. **SmartViz** also accesses the historical inference result data stored in the **DFB Elastic Search (ES)** database by making queries that are supported by the exposed REST API of the **DFB ES-proxy** service. **SmartViz** displays the inference results to the user through appropriate visualisations, as detailed in Section 5.5. **SmartViz** also allows the user to verify the displayed inference results and this verification information is fed back to the **DFB** by being published to a dedicated Kafka topic. The **DFB** then updates the relevant inference result entry in the **DFB ES** accordingly.

In parallel, a single instance of **StreamHandler** deployed at the Fog (**GRN F2**) receives anonymised AV data streams in real time from the **VideoAnony** instance. **StreamHandler** persistently stores the incoming anonymised AV data in a MinIO repository.

The **StreamHandler** instance also accesses the incoming inference results at the **DFB** in real time by subscribing to the Kafka topics where they are published by the **DatAna NiFi Cloud node. StreamHandler** uses the information in these results (e.g., AV source and time period of result) to generate AV files that correspond to each inference result.

Upon user demand, **SmartViz** requests data from **StreamHandler** that refer to a particular inference result by accessing a REST API exposed by **StreamHandler**. **StreamHandler** composes the AV data that is requested (if not already available) and returns the location of the corresponding AV data to **SmartViz**. **SmartViz** accesses the AV data and makes it available to the user for playback within its UI.

The **HDD** is deployed at the **PSNC HPC** at the Cloud and can receive the current Kafka topic partition configuration from the **DFB** on a periodic basis. After analysing it, the **HDD** returns a recommendation for an optimised Kafka topic partition configuration to the **DFB**.

Finally, **MARVdash** is orchestrating the deployment of all the implemented components across all infrastructure nodes (**GRN E1, GRN E3**, **GRN F2**, **PSNC HPC**).

Table 8 below presents a comprehensive list of components that were used in GRN1.

Table 8: MARVEL architectural components for GRN1 – Safer roads

MARVEL subsystem	Component information					
	Name	Owner	Functionality	Deployment		
				Instances	Layer	Infr. Node
Sensing and perception subsystem	AV Registry	ITML	Provides information on available AV sources	1	Fog	GRN F2
	Arduino Proxy	GRN	Transforms MQTT messages and relays over Serial protocol	1	Edge	GRN E3
	Arduino LED control	GRN	Aduino script for controlling LED board	1	Edge	Arduino Nano
Security, Privacy and data protection Subsystem	VideoAnony	FBK	Video anonymisation	1	Edge	GRN E1
	EdgeSec VPN	FORTH	Secure communications between services on infrastructure nodes	4	Edge, Fog, Cloud	GRN E1, GRN E3, GRN F2, PSNC HPC
Data management and distribution subsystem	DatAna	ATOS	Collection and transformation of raw inference results – MQTT brokers and NiFi nodes	3	Edge, Fog, Cloud	GRN E3, GRN F2, PSNC HPC
	DFB	ITML	Collection, persistent storage and fusion of SDM-compliant inference results – Kafka brokers, ES database	1	Cloud	PSNC HPC
	StreamHandler	INTRA	Persistent storage of AV data	1	Fog	GRN F2
	HDD	CNR	Kafka broker partition optimisation	1	Cloud	PSNC HPC
Audio, visual and multimodal AI Subsystem	YOLO-SED	AU, TAU	Anomaly detection using YOLO object detector and SED audio analysis	1	Cloud	GRN E3
Optimised E2F2C processing and deployment subsystem	MARVdash	FORTH	Service deployment and Kubernetes cluster management	1	Cloud	PSNC HPC
E2F2C infrastructure	HPC Infrastructure	PSNC	Cloud layer for GRN4 hosting VMs for service deployment	1	Cloud	PSNC HPC
	Management and orchestration of HPC resources	PSNC	Management and monitoring via SLURM software and OpenStack.	1	Cloud	PSNC HPC
User interactions and decision-making toolkit	SmartViz	ZELUS	UI and visualisations	1	Cloud	PSNC HPC

5.4.2 GRN2 – Road user behaviour: AI Inference runtime and deployment view

Figure 59 provides the deployment and runtime view of the MARVEL architecture for the GRN2 use case.

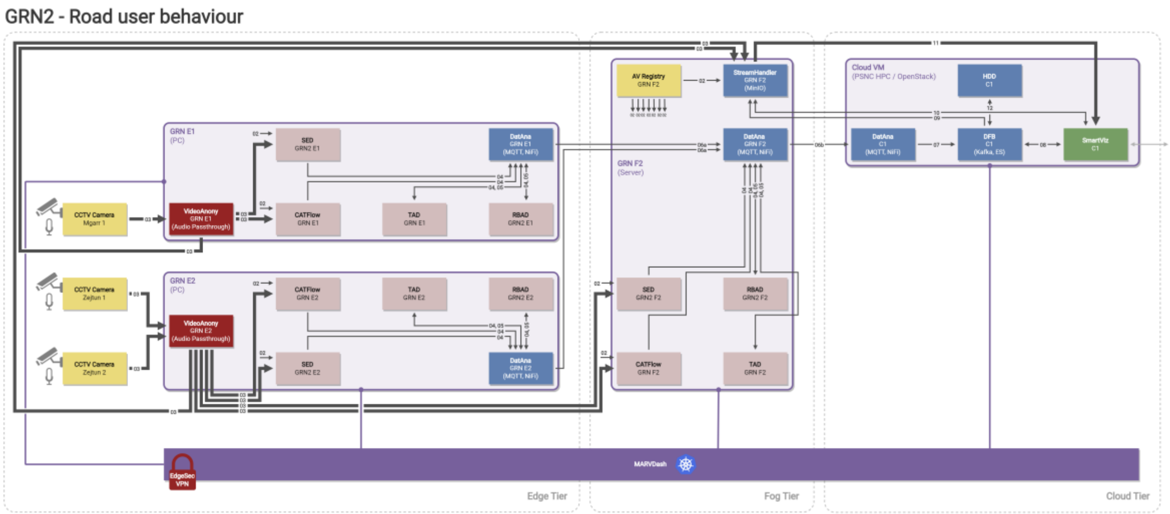


Figure 59: MARVEL R2 deployment and runtime view of the MARVEL architecture for GRN2 – Road user behaviour (*annotation label descriptions in Table 6, Section 5.2)*

This section describes the deployment, operation and interactions between components that are implemented for GRN2.

Audio and video streams are obtained from **three CCTV cameras** at two different locations in Malta (1 camera at Mgarr and 2 cameras at Zejtun).

A **VideoAnony** instance hosted on a PC at the Edge (**GRN E1)** at the Mgarr location receives the AV stream from the camera at Mgarr, while another **VideoAnony** instance hosted on a PC at the Edge (**GRN E2)** at the Zejtun location receives the AV streams from the two cameras at Zejtun.

During the system initialisation phase, the component instances that need to consume the produced anonymised AV data streams (i.e., **CATFlow**, **SED** and **StreamHandler**) request the metadata details of the **VideoAnony** AV sources from the **AV Registry** (**GRN F2**) via the corresponding REST API calls.

A **CATFlow** and a **SED** instance hosted on **GRN E1** receive the anonymised AV stream from the **VideoAnony** instance hosted on the same node. A second group of **CATFlow** and **SED** instances hosted on **GRN E2** receive one of the two anonymised AV streams from the **VideoAnony** instance hosted on the same node. A third group of **CATFlow** and **SED** instances hosted on the GRN Fog Server (**GRN F2**) receive the second anonymised AV stream from the **VideoAnony** instance hosted on **GRN E2**. Each **CATFlow** and **SED** instance receives the anonymised AV data stream from the corresponding **VideoAnony** instance in real time via RTSP. **CATFlow** instances analyse the stream's video section to produce the number of detected vehicles and traffic trajectories as inference results, while **SED** instances analyse the stream's audio section to produce raw inference results that refer to sound event detection.

Each **CATFlow** and **SED** instance publishes these inference results in real time to a dedicated topic (distinct for **CATFlow** and **SED**) of a **DatAna MQTT broker** residing at the same host device as the **CATFlow** instance (one MQTT broker on **GRN E1,** one MQTT broker on **GRN E2** and one MQTT broker on **GRN F2**).

At the same time, there is one **TAD** and one **RBAD** instance associated with each **CATFlow** instance that is deployed on the same infrastructure node (**GRN E1, GRN E2** and **GRN F2**). Each **TAD** and **RBAD** instance is subscribed to the topic of the corresponding **DatAna MQTT broker**, where **CATFlow** publishes its results in order to receive the output of the **CATFlow** instance it is associated with. Each **TAD** and **RBAD** instance processes the information it receives to detect anomalous events and produces its own raw inference results, which are published to dedicated topics (distinct for TAD and RBAD) on the **DatAna MQTT broker** that is deployed at the same node as the **TAD** and **RBAD** instance (**GRN E1, GRN E2** and **GRN F2**).

DatAna NiFi nodes collect the inference results from corresponding **DatAna MQTT brokers** by subscribing to the respective MQTT topics. Specifically, (i) the **DatAna NiFi Edge nodes** on **GRN E1** and on **GRN E2** receive the inference results that are published on the **DatAna MQTT Edge broker** on **GRN E1** and on **GRN E2** respectively and (ii) the **DatAna NiFi Fog node** on **GRN F2** receives the inference results that are published on the **DatAna MQTT Fog broker** on **GRN F2**. Each **DatAna NiFi node** performs certain operations on the collected raw inference results from all AI components to transform them to SDM-compliant data models (for more details, see Section 5.3.2). Then, each **DatAna NiFi node** pushes the transformed inference results to the **DatAna NiFi node** on the higher E/F/C layer, i.e., (i) the **DatAna NiFi Edge nodes** push results to the respective **DatAna NiFi Fog node** and (ii) the **DatAna NiFi Fog nodes** push results to the **DatAna NiFi Cloud node**.

The **DatAna NiFi Cloud node** publishes the transformed, SDM-compliant inference results it collects from all layers to dedicated Kafka topics at the **DFB** (a separate topic is used for the inference results of each AI component that produced them), which is deployed at the **PSNC HPC**. The **DFB** persistently stores the received results on the **DFB ES** repository. Furthermore, the **DFB** fuses similar consecutive inference results from **SED** that are very close in time to generate merged events that refer to longer periods.

SmartViz is deployed at the **PSNC HPC** and accesses the incoming inference results at the **DFB** in real time by subscribing to the Kafka topics where they are published by the **DatAna NiFi Cloud node**. **SmartViz** also accesses the historical inference result data stored in the **DFB Elastic Search (ES)** database by making queries that are supported by the exposed REST API of the **DFB ES-proxy** service. **SmartViz** displays the inference results to the user through appropriate visualisations, as detailed in Section 5.5. **SmartViz** also allows the user to verify the displayed inference results and this verification information is fed back to the **DFB** by being published to a dedicated Kafka topic. The **DFB** then updates the relevant inference result entry in the **DFB ES** accordingly.

In parallel, a single instance of **StreamHandler** deployed at the Fog (**GRN F2**) receives anonymised AV data streams in real time from the two **VideoAnony** instances. **StreamHandler** persistently stores the incoming anonymised AV data in a MinIO repository.

The **StreamHandler** instance also accesses the incoming inference results at the **DFB** in real time by subscribing to the Kafka topics where they are published by the **DatAna NiFi Cloud node. StreamHandler** uses the information in these results (e.g., AV source and time period of result) to generate AV files that correspond to each inference result.

Upon user demand, **SmartViz** requests data from **StreamHandler** that refer to a particular inference result by accessing a REST API exposed by **StreamHandler**. **StreamHandler** composes the AV data that is requested (if not already available) and returns the location of the corresponding AV data to **SmartViz**. **SmartViz** accesses the AV data and makes it available to the user for playback within its UI.

The **HDD** is deployed at the **PSNC HPC** in the Cloud and can receive the current Kafka topic partition configuration from the **DFB** on a periodic basis. After analysing it, the **HDD** returns a recommendation for an optimised Kafka topic partition configuration to the **DFB**.

Finally, **MARVdash** is orchestrating the deployment of all the implemented components across all infrastructure nodes (**GRN E1**, **GRN E2**, **GRN F2**, **PSNC HPC**).

Table 9 below presents a comprehensive list of components that were used in GRN2.

Table 9: MARVEL architectural components for GRN2 – Road user behaviour

MARVEL subsystem	Component information					
	Name	Owner	Functionality	Deployment		
				Instances	Layer	Infr. Node
Sensing and perception subsystem	AV Registry	ITML	Provides information on available AV sources	1	Fog	GRN F2
Security, Privacy and data protection Subsystem	VideoAnony	FBK	Video anonymisation	2	Edge	GRN E1, GRN E2
	EdgeSec VPN	FORTH	Secure communications between services on infrastructure nodes	4	Edge, Fog, Cloud	GRN E1, GRN E2, GRN F2, PSNC HPC
Data management and distribution subsystem	DatAna	ATOS	Collection and transformation of raw inference results – MQTT brokers and NiFi nodes	4	Edge, Fog, Cloud	GRN E1, GRN E2, GRN F2, PSNC HPC
	DFB	ITML	Collection, persistent storage and fusion of SDM-compliant inference results – Kafka brokers, ES database	1	Cloud	PSNC HPC
	StreamHandler	INTRA	Persistent storage of AV data	1	Fog	GRN F2
	HDD	CNR	Kafka broker partition optimisation	1	Cloud	PSNC HPC
Audio, visual and multimodal AI Subsystem	CATFlow	GRN	AI inference: vehicle count, traffic trajectories	3	Edge, Fog	GRN E1, GRN E2, GRN F2
	TAD	GRN	Text Anomaly Detection. Analysis of CATFlow raw inference results	3	Edge, Fog	GRN E1, GRN E2, GRN F2
	RBAD	AU	Rule-based Anomaly detection. Analysis of CATFlow raw inference results	3	Edge, Fog	GRN E1, GRN E2, GRN F2
	SED	TAU	Sound Event Detection in general audio signal	3	Cloud	GRN E1, GRN E2, GRN F2
Optimised E2F2C processing and deployment subsystem	MARVdash	FORTH	Service deployment and Kubernetes cluster management	1	Cloud	PSNC HPC
E2F2C infrastructure	HPC Infrastructure	PSNC	Cloud layer for GRN4 hosting VMs for service deployment	1	Cloud	PSNC HPC

	Management and orchestration of HPC resources	PSNC	Management and monitoring via SLURM software and OpenStack.	1	Cloud	PSNC HPC
User interactions and decision-making toolkit	SmartViz	ZELUS	UI and visualisations	1	Cloud	PSNC HPC

5.4.3 GRN3 – Traffic Anomalous Events: AI Inference runtime and deployment view

Figure 60 provides the deployment and runtime view of the MARVEL architecture for the GRN3 use case.

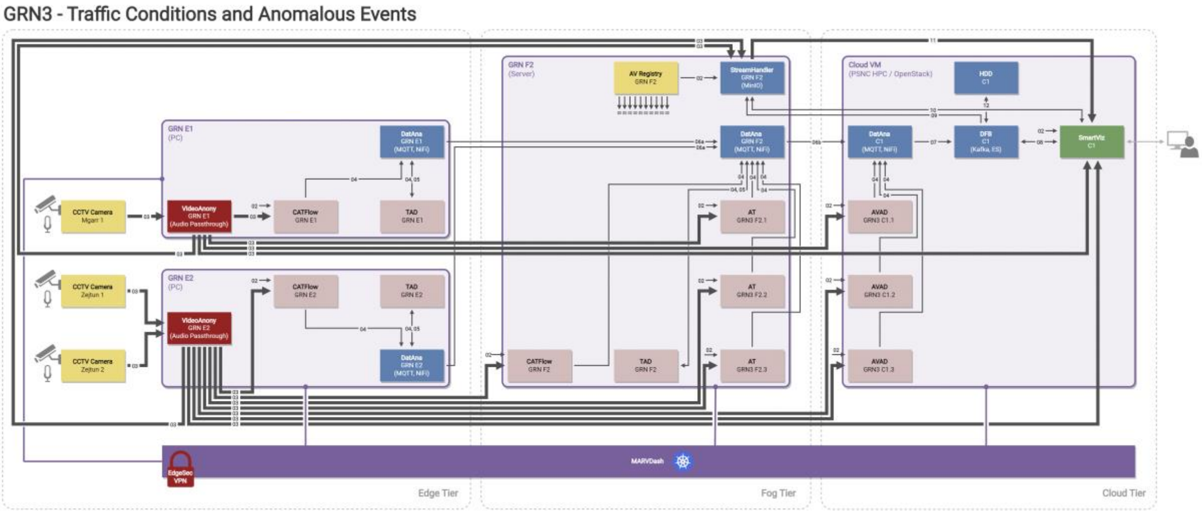


Figure 60: MARVEL R2 deployment and runtime view of the MARVEL architecture for GRN3: Traffic Conditions and Anomalous Events (*annotation label descriptions in Table 6, Section 5.2)*

This section describes the deployment, operation and interactions between components that are implemented for GRN3.

Audio and video streams are obtained from **three CCTV cameras** at two different locations in Malta (1 camera at Mgarr and 2 cameras at Zejtun).

A **VideoAnony** instance hosted on a PC at the Edge (**GRN E1)** at the Mgarr location receives the AV stream from the camera at Mgarr, while another **VideoAnony** instance hosted on a PC at the Edge (**GRN E2)** at the Zejtun location receives the AV streams from the two cameras at Zejtun.

During system initialisation, the component instances that need to consume the produced anonymised AV data streams (i.e., **CATFlow**, **AVAD**, **AT**, **StreamHandler** and **SmartViz**) request the metadata details of the **VideoAnony** AV sources from the **AV Registry** (**GRN F2**) via corresponding REST API calls.

A **CATFlow** instance hosted on **GRN E1** receives the anonymised AV stream from the **VideoAnony** instance hosted on the same node. A second **CATFlow** instance hosted on **GRN E2** receives one of the two anonymised AV streams from the **VideoAnony** instance hosted on the same node. A third **CATFlow** instance hosted on the GRN Fog Server (**GRN F2**) receives the second anonymised AV stream from the **VideoAnony** instance hosted on **GRN E2**. Each

CATFlow instance receives the anonymised AV data stream from the corresponding **VideoAnony** instance in real time via RTSP to analyse the stream's video section and produce the number of detected vehicles and traffic trajectories as inference results.

Each **CATFlow** instance publishes these inference results in real time to a dedicated topic of a **DatAna MQTT broker** residing at the same host device as the **CATFlow** instance (one MQTT broker on **GRN E1,** one MQTT broker on **GRN E2** and one MQTT broker on **GRN F2**).

At the same time, there is one **TAD** instance associated with each **CATFlow** instance that is deployed on the same infrastructure node (**GRN E1, GRN E2** and **GRN F2**). Each **TAD** instance is subscribed to the topic of the corresponding **DatAna MQTT broker**, where **CATFlow** publishes its results in order to receive the output of the **CATFlow** instance it is associated with. Each **TAD** instance processes the information it receives to detect anomalous events and produces its own raw inference results, which are published to another dedicated topic on the **DatAna MQTT broker** that is deployed at the same node as the **TAD** instance (**GRN E1, GRN E2** and **GRN F2**).

In parallel, there are three instances of **AT** deployed at the **GRN F2** and three instances of **AVAD** deployed at the **PSNC HPC**, which also receive the RTSP AV streams in real time from the two aforementioned **VideoAnony** instances (**VideoAnony** allows the audio to pass through it). **AVAD** processes both audio and video sections of the incoming stream to produce raw inference results that refer to anomaly detection, while **AT** processes the stream's audio section to produce inference results that refer to the activity of characteristic sounds inside audio segments. **AT** and **AVAD** instances publish their raw inference results in real time to dedicated topics of a **DatAna MQTT broker** residing at the **GRN F2** and the **PSNC HPC** respectively.

DatAna NiFi nodes collect the inference results from corresponding **DatAna MQTT brokers** by subscribing to the respective MQTT topics. Specifically, (i) the **DatAna NiFi Edge nodes** on **GRN E1** and on **GRN E2** receive the inference results that are published on the **DatAna MQTT Edge broker** on **GRN E1** and on **GRN E2** respectively, (ii) the **DatAna NiFi Fog node** on **GRN F2** receives the inference results that are published on the **DatAna MQTT Fog broker** on **GRN F2** and (iii) the **DatAna NiFi Cloud node** on the **PSNC HPC** receives the inference results that are published on the **DatAna MQTT Cloud broker** on the **PSNC HPC**. Each **DatAna NiFi node** performs certain operations on the collected raw inference results from all AI components to transform them to SDM-compliant data models (for more details, see Section 5.3.2). Then, each **DatAna NiFi node** pushes the transformed inference results to the **DatAna NiFi node** on the higher E/F/C layer, i.e., (i) the **DatAna NiFi Edge nodes** push results to the respective **DatAna NiFi Fog node** and (ii) the **DatAna NiFi Fog nodes** push results to the **DatAna NiFi Cloud node**.

The **DatAna NiFi Cloud node** publishes the transformed, SDM-compliant inference results it collects from all layers to dedicated Kafka topics at the **DFB** (a separate topic is used for the inference results of each AI component that produced them), which is deployed at the **PSNC HPC**. The **DFB** persistently stores the received results on the **DFB ES** repository. Furthermore, the **DFB** fuses similar consecutive inference results from **AVAD** and **AT** that are very close in time to generate merged events that refer to longer periods.

SmartViz is deployed at the **PSNC HPC** and accesses the incoming inference results at the **DFB** in real time by subscribing to the Kafka topics where they are published by the **DatAna NiFi Cloud node**. **SmartViz** also accesses the historical inference result data stored in the **DFB Elastic Search (ES)** database by making queries that are supported by the exposed REST API of the **DFB ES-proxy** service. **SmartViz** displays the inference results to the user through appropriate visualisations, as detailed in Section 5.5. **SmartViz** also allows the user to verify the displayed inference results and this verification information is fed back to the **DFB** by being published to a dedicated Kafka topic. The **DFB** then updates the relevant inference result entry in the **DFB ES** accordingly.

In parallel, a single instance of **StreamHandler** deployed at the Fog (**GRN F2**) receives anonymised AV data streams in real time from the two **VideoAnony** instances. **StreamHandler** persistently stores the incoming anonymised AV data in a MinIO repository.

The **StreamHandler** instance also accesses the incoming inference results at the **DFB** in real time by subscribing to the Kafka topics where they are published by the **DatAna NiFi Cloud node. StreamHandler** uses the information in these results (e.g., AV source and time period of result) to generate AV files that correspond to each inference result.

Upon user demand, **SmartViz** requests data from **StreamHandler** that refer to a particular inference result by accessing a REST API exposed by **StreamHandler**. **StreamHandler** composes the AV data that is requested (if not already available) and returns the location of the corresponding AV data to **SmartViz**. **SmartViz** accesses the AV data and makes it available to the user for playback within its UI.

Furthermore, upon user demand, **SmartViz** can connect to any of the two **VideoAnony** instances to receive a live AV data stream via RTSP and display it within its UI to the user. The connection is not continuous and is initiated only when a user requests a live AV feed from the location where an event has previously been detected.

The **HDD** is deployed at the **PSNC HPC** in the Cloud and can receive the current Kafka topic partition configuration from the **DFB** on a periodic basis. After analysing it, the **HDD** returns a recommendation for an optimised Kafka topic partition configuration to the **DFB**.

Finally, **MARVdash** is orchestrating the deployment of all the implemented components across all infrastructure nodes (**GRN E1**, **GRN E2**, **GRN F2**, **PSNC HPC**).

Table 10 below presents a comprehensive list of components that were used in GRN3.

Table 10: MARVEL architectural components for GRN3: Traffic Conditions and Anomalous Events

MARVEL subsystem	Component information					
	Name	Owner	Functionality	Deployment		
				Instances	Layer	Infr. Node
Sensing and perception subsystem	AV Registry	ITML	Provides information on available AV sources	1	Fog	GRN F2
Security, Privacy and data protection Subsystem	VideoAnony	FBK	Video anonymisation	2	Edge	GRN E1, GRN E2
	EdgeSec VPN	FORTH	Secure communications between services on infrastructure nodes	4	Edge, Fog, Cloud	GRN E1, GRN E2, GRN F2, PSNC HPC
Data management and distribution subsystem	DatAna	ATOS	Collection and transformation of raw inference results – MQTT brokers and NiFi nodes	4	Edge, Fog, Cloud	GRN E1, GRN E2, GRN F2, PSNC HPC
	DFB	ITML	Collection, persistent storage and fusion of SDM-compliant inference results – Kafka brokers, ES database	1	Cloud	PSNC HPC
	StreamHandler	INTRA	Persistent storage of AV data	1	Fog	GRN F2

	HDD	CNR	Kafka broker partition optimisation	1	Cloud	PSNC HPC
Audio, visual and multimodal AI Subsystem	CATFlow	GRN	AI inference: vehicle count, traffic trajectories	3	Edge, Fog	GRN E1, GRN E2, GRN F2
	TAD	GRN	Text Anomaly Detection. Analysis of CATFlow raw inference results	3	Edge, Fog	GRN E1, GRN E2, GRN F2
	AVAD	AU	Audio-Visual Anomaly detection. Detecting deviations from normality within video frames and corresponding scene audio	3	Cloud	PSNC HPC
	AT	TAU	Audio Tagging in fixed length segments	3	Fog	GRN F2
Optimised E2F2C processing and deployment subsystem	MARVdash	FORTH	Service deployment and Kubernetes cluster management	1	Cloud	PSNC HPC
E2F2C infrastructure	HPC Infrastructure	PSNC	Cloud layer for GRN4 hosting VMs for service deployment	1	Cloud	PSNC HPC
	Management and orchestration of HPC resources	PSNC	Management and monitoring via SLURM software and OpenStack.	1	Cloud	PSNC HPC
User interactions and decision-making toolkit	SmartViz	ZELUS	UI and visualisations	1	Cloud	PSNC HPC

5.4.4 GRN4 – Junction Traffic Trajectory: AI Inference runtime and deployment view

Figure 61 provides the deployment and runtime view of the MARVEL architecture for the GRN4 use case.

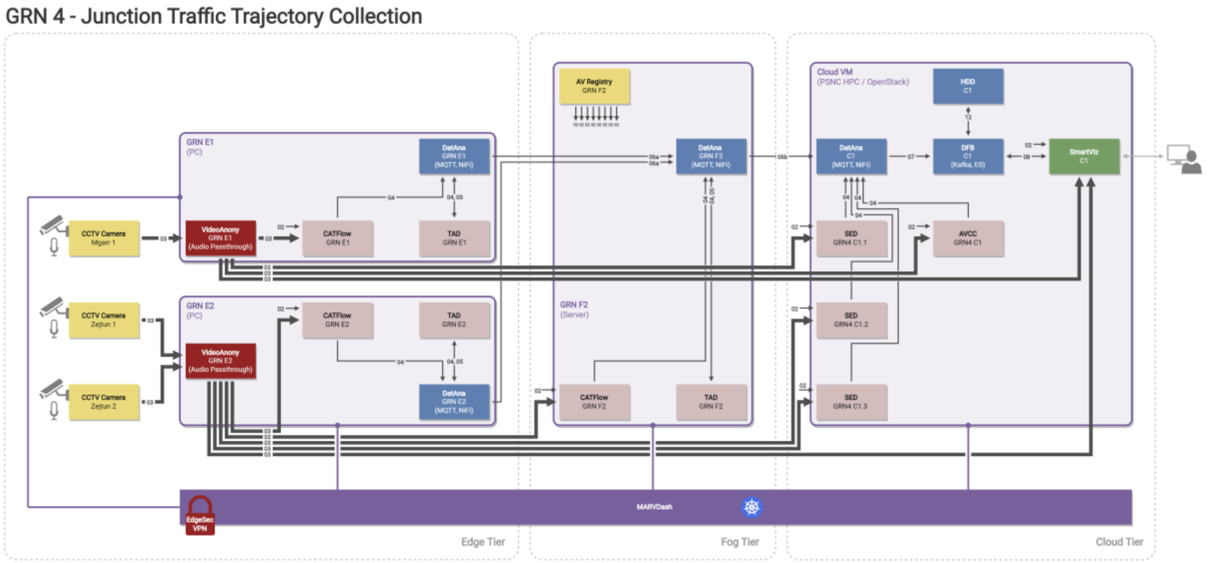


Figure 61: MARVEL R2 deployment and runtime view of the MARVEL architecture for GRN4: Junction Traffic Trajectory Collection (*annotation label descriptions in Table 6, Section 5.2)*

This section describes the deployment, operation and interactions between components that are implemented for GRN4.

Audio and video streams are obtained from **three CCTV cameras** at two different locations in Malta (1 camera at Mgarr and 2 cameras at Zejtun).

A **VideoAnony** instance hosted on a PC at the Edge (**GRN E1)** at the Mgarr location receives the AV stream from the camera at Mgarr, while another **VideoAnony** instance hosted on a PC at the Edge (**GRN E2)** at the Zejtun location receives the AV streams from the two cameras at Zejtun.

During system initialisation, the component instances that need to consume the produced anonymised AV data streams (i.e., **CATFlow**, **SED**, **AVCC** and **SmartViz**) request the metadata details of the **VideoAnony** AV sources from the **AV Registry** (**GRN F2**) via corresponding REST API calls.

A **CATFlow** instance hosted on **GRN E1** receives the anonymised AV stream from the **VideoAnony** instance hosted on the same node. A second **CATFlow** instance hosted on **GRN E2** receives one of the two anonymised AV streams from the **VideoAnony** instance hosted on the same node. A third **CATFlow** instance hosted on the GRN Fog Server (**GRN F2**) receives the second anonymised AV stream from the **VideoAnony** instance hosted on **GRN E2**. Each **CATFlow** instance receives the anonymised AV data stream from the corresponding **VideoAnony** instance in real time via RTSP to analyse the stream's video section and produce the number of detected vehicles and traffic trajectories as inference results.

Each **CATFlow** instance publishes these inference results in real time to a dedicated topic of a **DatAna MQTT broker** residing at the same host device as the **CATFlow** instance (one MQTT broker on **GRN E1,** one MQTT broker on **GRN E2** and one MQTT broker on **GRN F2**).

At the same time, there is one **TAD** instance associated with each **CATFlow** instance that is deployed on the same infrastructure node (**GRN E1, GRN E2** and **GRN F2**). Each **TAD** instance is subscribed to the topic of the corresponding **DatAna MQTT broker**, where **CATFlow** publishes its results in order to receive the output of the **CATFlow** instance it is associated with. Each **TAD** instance processes the information it receives to detect anomalous events and produces its own raw inference results, which are published to another dedicated topic on the **DatAna MQTT broker** that is deployed at the same node as the **TAD** instance (**GRN E1, GRN E2** and **GRN F2**).

In parallel, there are three instances of **SED** deployed at the **PSNC HPC**, which also receive the RTSP AV streams in real time from the two **VideoAnony** instances (**VideoAnony** allows the audio to pass through it). There is also one instance of **AVCC** deployed at the **PSNC HPC**, which receives the RTSP AV stream in real time from one of the **VideoAnony** instances (from **GRN E1**). **SED** processes the audio sections of the incoming stream to produce raw inference results that refer to sound event detection, while **AVCC** processes the stream's audio and video sections to produce inference results that refer to crowd counting and the generation of visual heatmaps. All **SED** and **AVCC** instances publish their raw inference results in real time to dedicated topics of a **DatAna MQTT broker** residing at the **PSNC HPC**.

DatAna NiFi nodes collect the inference results from corresponding **DatAna MQTT brokers** by subscribing to the respective MQTT topics. Specifically, (i) the **DatAna NiFi Edge nodes** on **GRN E1** and on **GRN E2** receive the inference results that are published on the **DatAna MQTT Edge broker** on **GRN E1** and on **GRN E2** respectively, (ii) the **DatAna NiFi Fog node** on **GRN F2** receives the inference results that are published on the **DatAna MQTT Fog broker** on **GRN F2** and (iii) the **DatAna NiFi Cloud node** on the **PSNC HPC** receives the inference results that are published on the **DatAna MQTT Cloud broker** on the **PSNC HPC**. Each **DatAna NiFi node** performs certain operations on the collected raw inference results from all AI components to transform them to SDM-compliant data models (for more details, see Section 5.3.2). Then, each **DatAna NiFi node** pushes the transformed inference results to the **DatAna NiFi node** on the higher E/F/C layer, i.e., (i) the **DatAna NiFi Edge nodes** push results to the respective **DatAna NiFi Fog node** and (ii) the **DatAna NiFi Fog nodes** push results to the **DatAna NiFi Cloud node**.

The **DatAna NiFi Cloud node** publishes the transformed, SDM-compliant inference results it collects from all layers to dedicated Kafka topics at the **DFB** (a separate topic is used for the inference results of each AI component that produced them), which is deployed at the **PSNC HPC**. The **DFB** persistently stores the received results on the **DFB ES** repository. Furthermore, the **DFB** fuses similar consecutive inference results from **SED** that are very close in time to generate merged events that refer to longer periods.

SmartViz is deployed at the **PSNC HPC** and accesses the incoming inference results at the **DFB** in real time by subscribing to the Kafka topics where they are published by the **DatAna NiFi Cloud node**. **SmartViz** also accesses the historical inference result data stored in the **DFB Elastic Search (ES)** database by making queries that are supported by the exposed REST API of the **DFB ES-proxy** service. **SmartViz** displays the inference results to the user through appropriate visualisations, as detailed in Section 5.5. **SmartViz** also allows the user to verify the displayed inference results and this verification information is fed back to the **DFB** by being published to a dedicated Kafka topic. The **DFB** then updates the relevant inference result entry in the **DFB ES** accordingly.

Furthermore, upon user demand, **SmartViz** can connect to any of the two **VideoAnony** instances to receive a live AV data stream via RTSP and display it within its UI to the user. The connection is not continuous and is initiated only when a user requests a live AV feed from the location where an event has previously been detected.

The **HDD** is deployed at the **PSNC HPC** in the Cloud and can receive the current Kafka topic partition configuration from the **DFB** on a periodic basis. After analysing it, the **HDD** returns a recommendation for an optimised Kafka topic partition configuration to the **DFB**.

Finally, **MARVdash** is orchestrating the deployment of all the implemented components across all infrastructure nodes (**GRN E1**, **GRN E2**, **GRN F2**, **PSNC HPC**).

Table 11 below presents a comprehensive list of components that were used in GRN4.

Table 11: MARVEL architectural components for GRN4: Junction Traffic Trajectory Collection

MARVEL subsystem	Component information					
	Name	Owner	Functionality	Deployment		
				Instances	Layer	Infr. Node
Sensing and perception subsystem	AV Registry	ITML	Provides information on available AV sources	1	Fog	GRN F2
Security, Privacy and data protection Subsystem	VideoAnony	FBK	Video anonymisation	2	Edge	GRN E1, GRN E2
	EdgeSec VPN	FORTH	Secure communications between services on infrastructure nodes	4	Edge, Fog, Cloud	GRN E1, GRN E2, GRN F2, PSNC HPC
Data management and distribution subsystem	DatAna	ATOS	Collection and transformation of raw inference results – MQTT brokers and NiFi nodes	4	Edge, Fog, Cloud	GRN E1, GRN E2, GRN F2, PSNC HPC
	DFB	ITML	Collection, persistent storage and fusion of SDM-compliant inference results – Kafka brokers, ES database	1	Cloud	PSNC HPC
	HDD	CNR	Kafka broker partition optimisation	1	Cloud	PSNC HPC
Audio, visual and multimodal AI Subsystem	CATFlow	GRN	AI inference: vehicle count, traffic trajectories	3	Edge, Fog	GRN E1, GRN E2, GRN F2
	TAD	GRN	Text Anomaly Detection. Analysis of CATFlow raw inference results	3	Edge, Fog	GRN E1, GRN E2, GRN F2
	SED	TAU	Sound Event Detection in general audio signal	3	Cloud	PSNC HPC
	AVCC	AU	Audio-Visual Crowd Counting. Total number of people present in given video frames and from the corresponding ambient audio.	1	Cloud	PSNC HPC
Optimised E2F2C processing and deployment subsystem	MARVdash	FORTH	Service deployment and Kubernetes cluster management	1	Cloud	PSNC HPC
E2F2C infrastructure	HPC Infrastructure	PSNC	Cloud layer for GRN4 hosting VMs for service deployment	1	Cloud	PSNC HPC
	Management and orchestration of HPC resources	PSNC	Management and monitoring via SLURM software and OpenStack.	1	Cloud	PSNC HPC

User interactions and decision-making toolkit	SmartViz	ZELUS	UI and visualisations	1	Cloud	PSNC HPC

5.4.5 MT1 – Monitoring of crowded areas: AI Inference runtime and deployment view

Figure 62 provides the deployment and runtime view of the MARVEL architecture for the MT1 use case.

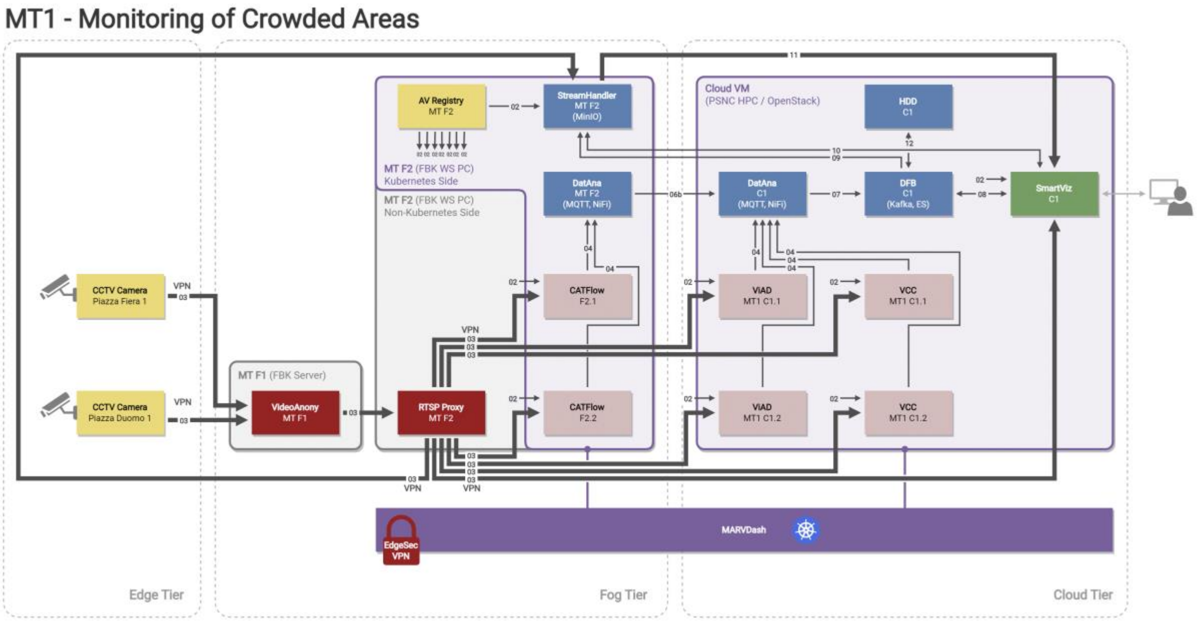


Figure 62: MARVEL R2 deployment and runtime view of the MARVEL architecture for MT1: Monitoring of crowded areas (*annotation label descriptions in Table 6, Section 5.2)*

This section describes the deployment, operation and interactions between components that are implemented for MT1.

Video streams are obtained from **two CCTV cameras** at two different public locations in the municipality of Trento (1 camera at Piazza Fiera and 1 camera at Piazza Duomo).

Due to restrictions associated with data privacy regulations and network security concerns expressed by the Municipality of Trento (MT), a customised solution was implemented to allow MARVEL components to access the AV data streams of the **MT1 CCTV cameras**. According to data privacy regulations, access to the raw AV data feeds from the CCTV cameras that are part of the MT network is restricted only to authorised personnel and to trusted devices from within the network. Therefore, it was not possible to provide MARVEL components with direct access to the raw AV data, but only to anonymised versions of it [59]. Following deliberations with the MT authorities and based on an agreement that nominates FBK as a data processor under certain constraints, a solution was reached that involved the use of an external server (**MT F1**), which is managed by FBK. However, this server (**MT F1**) could not be attached to

[59] This issue could be overcome by processing the raw AV data streams with VideoAnony before feeding them to other MARVEL components. However, there were additional issues related to network security, since (i) no device of the MT network could be attached to the MARVEL Kubernetes cluster and (ii) only trusted external endpoints with compatible network security policies can be granted access to the MT network, where the CCTV cameras are hosted.

the MARVEL Kubernetes cluster, because such an action would violate the conditions and security policies foreseen by the agreement between MT and FBK. Therefore, it was necessary to deploy **VideoAnony** services directly on the **MT F1** server at the Fog, without them being hosted on the MARVEL Kubernetes cluster. These **VideoAnony** services could consume the raw AV streams through a VPN tunnel. In addition, a second Fog server was set up at FBK (**MT F2**), which was allowed to (i) be attached to the MARVEL Kubernetes cluster and (ii) gain access to the **VideoAnony** services hosted on the first infrastructure node of FBK (**MT F1**) through an **RTSP Proxy** service that was not part of the Kubernetes cluster, but could re-stream the output of **VideoAnony** towards the Kubernetes cluster using a VPN tunnel. Therefore, using this configuration, it was possible for all components that are hosted within the MARVEL Kubernetes cluster to gain access to anonymised AV data streams produced by VideoAnony at **MT F1**.

A **VideoAnony** instance is deployed on **MT F1** and is tasked with processing the streams the two CCTV cameras. The output is exposed to an **RTSP Proxy** service that is deployed on **MT F2** outside of the MARVEL Kubernetes cluster via RTSP. This **RTSP Proxy** service can expose the anonymised streams from the VideoAnony service to components residing within the MARVEL Kubernetes cluster.

During the system initialisation phase, the component instances that need to consume the produced anonymised AV data streams (i.e., **CATFlow**, **ViAD**, **VCC**, **StreamHandler** and **SmartViz**) request the metadata details of the **VideoAnony** AV sources from the **AV Registry** (**MT F2**) via corresponding REST API calls.

Two **CATFlow** instances are deployed at the Fog layer on **MT F2** and receive the output of anonymised AV data stream in real time from the **VideoAnony** instance on **MT F1** through the **RTSP Proxy** service on **MT F2** via RTSP to analyse the stream's video section and produce the number of detected vehicles and traffic trajectories as inference results.

Both **CATFlow** instances publish their inference results in real time to a dedicated topic of the **DatAna MQTT broker** residing at **MT F2**.

In parallel, there are two instances of **ViAD** and two instances of **VCC** deployed at the **PSNC HPC**, which also receive the RTSP AV streams from the **VideoAnony** instance on **MT F1** through the **RTSP Proxy** service on **MT F2**. **ViAD** and **VCC** process the video section of the incoming streams to produce raw inference results that refer to anomaly detection and to crowd counting and generation of visual heatmaps respectively. All **ViAD** and **VCC** instances publish these raw inference results in real time to dedicated topics of a **DatAna MQTT broker** residing at the **PSNC HPC**.

DatAna NiFi nodes collect the inference results from corresponding **DatAna MQTT brokers** by subscribing to the respective MQTT topics. Specifically, (i) the **DatAna NiFi Fog node** on **MT F2** receives the inference results that are published on the **DatAna MQTT Fog broker** on **MT F2** and (ii) the **DatAna NiFi Cloud node** on the **PSNC HPC** receives the inference results that are published on the **DatAna MQTT Cloud broker** on the **PSNC HPC**. Each **DatAna NiFi node** performs certain operations on the collected raw inference results from all AI components to transform them to SDM-compliant data models (for more details, see Section 5.3.2). Then, each **DatAna NiFi node** pushes the transformed inference results to the **DatAna NiFi node** on the higher E/F/C layer, i.e., the **DatAna NiFi Fog node** pushes results to the **DatAna NiFi Cloud node**.

The **DatAna NiFi Cloud node** publishes the transformed inference results it collects from all layers to dedicated Kafka topics at the **DFB** (a separate topic is used for the inference results of each AI component that produced them), which is deployed at the **PSNC HPC**. The **DFB** persistently stores the received results on the **DFB ES** repository. Furthermore, the **DFB** fuses similar consecutive inference results from **ViAD** that are very close in time to generate merged events that refer to longer periods.

SmartViz is deployed at the **PSNC HPC** and accesses the incoming inference results at the **DFB** in real time by subscribing to the Kafka topics where they are published by the **DatAna NiFi Cloud node**. **SmartViz** also accesses the historical inference result data stored in the **DFB ES** database by making queries that are supported by the exposed REST API of the **DFB ES-proxy** service. **SmartViz** displays the inference results to the user through appropriate visualisations, as detailed in Section 5.5. **SmartViz** also allows the user to verify the displayed inference results and this verification information is fed back to the **DFB** by being published to a dedicated Kafka topic. The **DFB** then updates the relevant inference result entry in the **DFB ES** accordingly.

In parallel, a single instance of **StreamHandler** deployed at the Fog (**MT F2**) receives anonymised AV data streams in real time from the **VideoAnony** instance on **MT F1** through the **RTSP Proxy** service on **MT F2**. **StreamHandler** persistently stores the incoming anonymised AV data in a MinIO repository.

The **StreamHandler** instance also accesses the incoming inference results at the **DFB** in real time by subscribing to the Kafka topics where they are published by the **DatAna NiFi Cloud node. StreamHandler** uses the information in these results (e.g., AV source and time period of result) to generate AV files that correspond to each inference result.

Upon user demand, **SmartViz** requests data from **StreamHandler** that refer to a particular inference result by accessing a REST API exposed by **StreamHandler**. **StreamHandler** composes the AV data that is requested (if not already available) and returns the location of the corresponding AV data to **SmartViz**. **SmartViz** accesses the AV data and makes it available to the user for playback within its UI.

Furthermore, upon user demand, **SmartViz** can connect to the **VideoAnony** instance on **MT F1** through the **RTSP Proxy** service on **MT F2** to receive a live AV data stream via RTSP and display it within its UI to the user. The connection is not continuous and is initiated only when a user requests a live AV feed from the location where an event has previously been detected.

The **HDD** is deployed at the **PSNC HPC** at the Cloud and can receive the current Kafka topic partition configuration from the **DFB** on a periodic basis. After analysing it, the **HDD** returns a recommendation for an optimised Kafka topic partition configuration to the **DFB**.

Finally, **MARVdash** is orchestrating the deployment of all the implemented components across the infrastructure nodes of **MT F2** and **PSNC HPC**.

Table 12 below presents a comprehensive list of components that were used in MT1.

Table 12: MARVEL architectural components for MT1: Monitoring of crowded areas

MARVEL subsystem	Component information					
	Name	Owner	Functionality	Deployment		
				Instances	Layer	Infr. Node
Sensing and perception subsystem	AV Registry	ITML	Provides information on available AV sources	1	Fog	MT F2
Security, Privacy and	VideoAnony	FBK	Video anonymisation	1	Fog	MT F1

data protection Subsystem	EdgeSec VPN	FORTH	Secure communications between services on infrastructure nodes	2	Fog, Cloud	MT F2, PSNC HPC
Data management and distribution subsystem	DatAna	ATOS	Collection and transformation of raw inference results – MQTT brokers and NiFi nodes	2	Fog, Cloud	MT F2, PSNC HPC
	DFB	ITML	Collection, persistent storage and fusion of SDM-compliant inference results – Kafka brokers, ES database	1	Cloud	PSNC HPC
	StreamHandler	INTRA	Persistent storage of AV data	1	Fog	MT F2
	HDD	CNR	Kafka broker partition optimisation	1	Cloud	PSNC HPC
Audio, visual and multimodal AI Subsystem	CATFlow	GRN	AI inference: pedestrian count, traffic trajectories	2	Fog	MT F2
	ViAD	AU	Visual Anomaly detection. Detecting deviations from normality within video frames	2	Cloud	PSNC HPC
	VCC	AU	Visual Crowd Counting. Total number of people present in given video frames.	2	Cloud	PSNC HPC
Optimised E2F2C processing and deployment subsystem	MARVdash	FORTH	Service deployment and Kubernetes cluster management	1	Cloud	PSNC HPC
E2F2C infrastructure	HPC Infrastructure	PSNC	Cloud layer for GRN4 hosting VMs for service deployment	1	Cloud	PSNC HPC
	Management and orchestration of HPC resources	PSNC	Management and monitoring via SLURM software and OpenStack.	1	Cloud	PSNC HPC
User interactions and decision-making toolkit	SmartViz	ZELUS	UI and visualisations	1	Cloud	PSNC HPC

5.4.6 MT2 – Detecting criminal/anti-social behaviours: AI Inference runtime and deployment view

Figure 64 provides the deployment and runtime view of the MARVEL architecture for the MT3 use case.

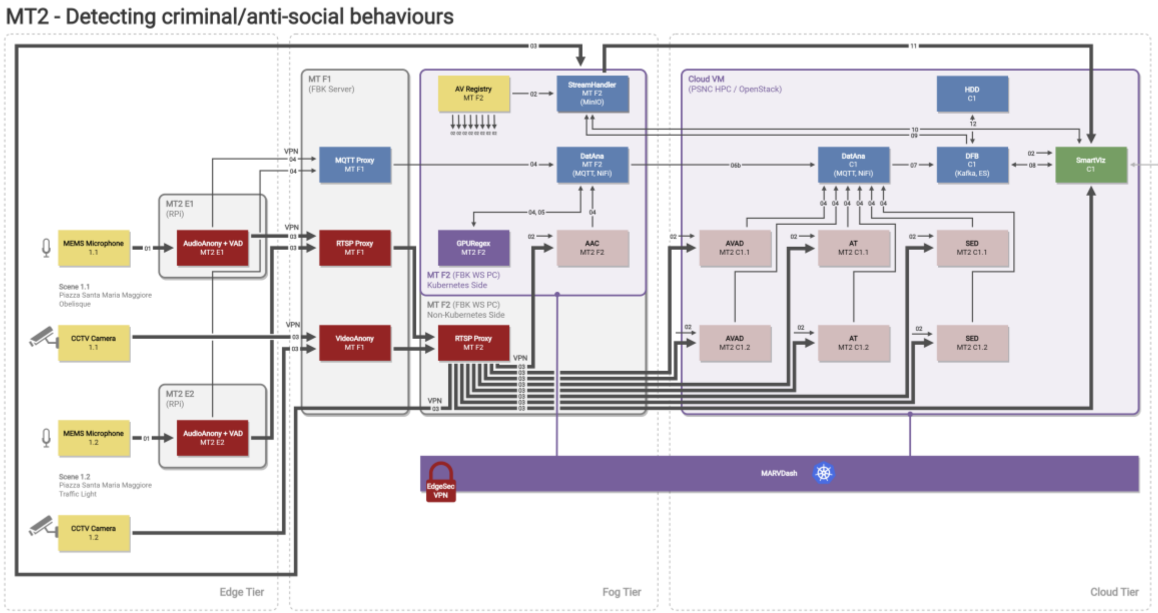


Figure 63: MARVEL R2 deployment and runtime view of the MARVEL architecture for MT2 – Detecting criminal/anti-social behaviours (*annotation label descriptions in Table 6, Section 5.2)*

This section describes the deployment, operation and interactions between components that are implemented for MT2.

Audio streams are obtained from two **MEMS microphones** that are hosted on Raspberry Pi devices connected to the network of the Municipality of Trento and located at the Piazza Santa Maria Maggiore in Trento. Video streams are obtained from two **CCTV cameras** at the same location.

Due to restrictions associated with data privacy regulations and network security concerns expressed by the Municipality of Trento (MT), a customised solution was implemented to allow MARVEL components to access the AV data streams of the **MT2 MEMS microphones** and **CCTV cameras** (similar to the solution implemented for MT1, but additionally addressing the needs of anonymising audio at the edge within the MT network). According to data privacy regulations, access to the raw AV data feeds (microphone and CCTV camera) that are part of the MT network is restricted only to authorised personnel and to trusted devices within the network. Therefore, it was not possible to provide MARVEL components with direct access to the raw AV data, but only to anonymised versions of it [60]. Following deliberations with the MT authorities and based on an agreement that nominates FBK as a data processor under certain constraints, a solution was reached that involved the use of an external server (**MT F1**), which is managed by FBK. However, this server (**MT F1**) could not be attached to the MARVEL Kubernetes cluster, because such an action would violate the conditions and security policies foreseen by the agreement between MT and FBK.

Regarding audio, it was decided to deploy **AudioAnony** (coupled with **VAD**) directly on the **MT2 E1 and MT2 E2 Raspberry Pi devices** in order to provide anonymisation at the Edge,

[60] This issue could be overcome by processing the raw AV data streams with AudioAnony and VideoAnony before feeding them to other MARVEL components. However, there were additional issues related to network security, since (i) no device that belonged in the MT network could be attached to the MARVEL Kubernetes cluster and (ii) only trusted external endpoints with compatible network security policies can be granted access to the MT network, where the Raspberry Pi (MT E1) with the MEMS microphone and the CCTV cameras are hosted.

considering that the RPi could provide sufficient computational power for this service. However, since **MT2 E1** and **MT2 E2** were part of the MT network, they could not be attached as nodes to the MARVEL Kubernetes cluster. Furthermore, due to the agreement between MT and FBK, **MT2 E1** and **MT2 E2** could only be accessed from **MT F1** using VPN. In order to grant access to the anonymised audio from **AudioAnony** on **MT2 E1** and on **MT2 E2** to other endpoints within the MARVEL Kubernetes cluster, an **RTSP Proxy F1** service was deployed directly on **MT F1** that could re-stream the output audio stream from **AudioAnony** on **MT E1** and on **MT2 E2**.

Regarding video, it was necessary to deploy a **VideoAnony** service directly on the **MT F1** server at the Fog, without it being hosted on the MARVEL Kubernetes cluster. This **VideoAnony** service could consume the raw AV streams from the CCTV cameras through a VPN tunnel.

In addition, a second Fog server was set up at FBK (**MT F2**), which was allowed to (i) be attached to the MARVEL Kubernetes cluster and (ii) gain access to the **RTSP Proxy F1** and **VideoAnony** services hosted on the first infrastructure node of FBK (**MT F1**) through an **RTSP Proxy F2** service that was not part of the Kubernetes cluster, but could re-stream the output of **AudioAnony** and **VideoAnony** towards the Kubernetes cluster using a VPN tunnel. Therefore, using this configuration, it was possible for all components that are hosted within the MARVEL Kubernetes cluster to gain access to anonymised AV data streams produced by **AudioAnony** on **MT2 E1** and on **MT2 E2** and by **VideoAnony** on **MT F1**.

AudioAnony instances on **MT2 E1** and on **MT2 E2** are coupled with **VAD** instances within the same containers. **VAD** accesses the raw audio stream and detects voice activity boundaries, which are directly provided to **AudioAnony** for controlling the activation of the anonymisation process. The **VAD** component within the **AudioAnony** container on **MT2 E1** and on **MT2 E2** also needs to publish the voice activity boundaries it detects as raw inference results in real time to a dedicated topic of the **DatAna MQTT Proxy** broker residing at **MT F2** (within Kubernetes). For this purpose, an **MQTT Proxy** service was deployed on **MT F1** to which **VAD** could publish MQTT messages via VPN, which were then restreamed through another VPN tunnel towards the **DatAna MQTT broker on MT F2** within the Kubernetes cluster.

During system initialisation, the component instances that need to consume the produced anonymised AV data streams (i.e., **AAC, AVAD, AT**, **SED**, **StreamHandler** and **SmartViz**) request the metadata details of the **RTSP Proxy F2** AV sources (AudioAnony and VideoAnony) from the **AV Registry** (**MT F2**) via corresponding REST API calls.

An instance of **AAC** is deployed on **MT F2** and receives the output of anonymised AV data stream in real time from the **AudioAnony** instance on **MT2 E1** through the **RTSP Proxy** service on **MT F2** via RTSP to analyse the stream’s audio and produce inference results comprising audio captions.

The **AAC** instance publishes these inference results in real time to a dedicated topic of a **DatAna MQTT broker** residing at **MT F2**.

At the same time, there is a **GPURegex** instance deployed on **MT F2** associated with the **AAC** instance on **MT F2.** The **GPURegex** instance is subscribed to the topic of the corresponding **DatAna MQTT broker** on **MT F2**, where **AAC** publishes its results in order to receive its output. **GPURegex** processes the information it receives to detect criminal and anti-social actions by searching for specific keywords and produces its own raw inference results (alerts), which are published to another dedicated topic on the **DatAna MQTT broker** on **MT F2**.

In parallel, there instances of **AT**, **SED** and **AVAD** (two of each) deployed at the **PSNC HPC**. All these components (**AT, SED, AVAD**) receive the two audio RTSP streams from **AudioAnony instances** on **MT2 E1** and on **MT2 E2** in real time through the **RTSP Proxy** on **MT F2**. **AVAD** also receives the two RTSP AV streams in real time from the **VideoAnony** instance on **MT F1** through the **RTSP Proxy** on **MT F2. AT** and **SED** process the incoming audio data stream to produce raw inference results that refer to the activity of characteristic sounds inside audio segments and to detected sound events respectively. **AVAD** processes the incoming audio and video data streams to produce raw inference results that refer to anomaly detection. All **AT**, **SED** and **AVAD** instances publish their raw inference results in real time to dedicated topics of the **DatAna MQTT broker** residing at the **PSNC HPC**.

DatAna NiFi nodes collect the inference results from corresponding **DatAna MQTT brokers** by subscribing to the respective MQTT topics. Specifically, (i) the **DatAna NiFi Fog node** on **MT F2** receives the inference results that are published on the **DatAna MQTT Fog broker** on **MT F2** and (ii) the **DatAna NiFi Cloud node** on the **PSNC HPC** receives the inference results that are published on the **DatAna MQTT Cloud broker** on the **PSNC HPC**. Each **DatAna NiFi node** performs certain operations on the collected raw inference results from all AI components to transform them to SDM-compliant data models (for more details, see Section 5.3.2). Then, each **DatAna NiFi node** pushes the transformed inference results to the **DatAna NiFi node** on the higher E/F/C layer, i.e., the **DatAna NiFi Fog node** pushes results to the **DatAna NiFi Cloud node**.

The **DatAna NiFi Cloud node** publishes the transformed inference results it collects from all layers to dedicated Kafka topics at the **DFB** (a separate topic is used for the inference results of each AI component that produced them), which is deployed at the **PSNC HPC**. The **DFB** persistently stores the received results on the **DFB ES** repository. Furthermore, the **DFB** fuses similar consecutive inference results from **AVAD, SED** and **AT** that are very close in time to generate merged events that refer to longer periods.

SmartViz is deployed at the **PSNC HPC** and accesses the incoming inference results at the **DFB** in real time by subscribing to the Kafka topics where they are published by the **DatAna NiFi Cloud node**. **SmartViz** also accesses the historical inference result data stored in the **DFB ES** database by making queries that are supported by the exposed REST API of the **DFB ES-proxy** service. **SmartViz** displays the inference results to the user through appropriate visualisations, as detailed in Section 5.5. **SmartViz** also allows the user to verify the displayed inference results and this verification information is fed back to the **DFB** by being published to a dedicated Kafka topic. The **DFB** then updates the relevant inference result entry in the **DFB ES** accordingly.

In parallel, a single instance of **StreamHandler** deployed at the Fog (**MT F2**) receives anonymised AV data streams in real time from the **RTSP Proxy** service on **MT F2**. **StreamHandler** persistently stores the incoming anonymised AV data in a MinIO repository.

The **StreamHandler** instance also accesses the incoming inference results at the **DFB** in real time by subscribing to the Kafka topics where they are published by the **DatAna NiFi Cloud node. StreamHandler** uses the information in these results (e.g., AV source and time period of result) to generate AV files that correspond to each inference result.

Upon user demand, **SmartViz** requests data from **StreamHandler** that refer to a particular inference result by accessing a REST API exposed by **StreamHandler**. **StreamHandler** composes the AV data that is requested (if not already available) and returns the location of the corresponding AV data to **SmartViz**. **SmartViz** accesses the AV data and makes it available to the user for playback within its UI.

Furthermore, upon user demand, **SmartViz** can connect to the **RTSP Proxy F2** and receive a live anonymised AV data stream via RTSP from **AudioAnony** on **MT2 E1** or on **MT2 E2** or from **VideoAnony** on **MT F1** and display it within its UI to the user. The connection is not continuous and is initiated only when a user requests a live AV feed from the location where an event has previously been detected.

The **HDD** is deployed at the **PSNC HPC** in the Cloud and can receive the current Kafka topic partition configuration from the **DFB** on a periodic basis. After analysing it, the **HDD** returns a recommendation for an optimised Kafka topic partition configuration to the **DFB**.

Finally, **MARVdash** is orchestrating the deployment of all the implemented components across the infrastructure nodes of **MT F2** and **PSNC HPC**.

Table 13 provides the list of components together with their main functionalities, across MARVEL functional subsystems that were used in MT3.

Table 13: MARVEL architectural components for MT2 – Detecting criminal/anti-social behaviours

MARVEL subsystem	Component information					
	Name	Owner	Functionality	Deployment		
				Instances	Layer	Infr. Node
Sensing and perception subsystem	MEMS microphone	IFAG	Audio data acquisition via IM69D130 and Edge processing of the acquired data	2	Edge	MT2 E1, MT2 E2
	AV Registry	ITML	Provides information on available AV sources	1	Fog	MT F2
Security, Privacy and data protection Subsystem	AudioAnony	FBK	Audio anonymisation	2	Edge	MT2 E1, MT2 E2
	VAD	AUD	Voice Activity Detection detects voiced segments either in batched or online fashion.	2	Edge	MT2 E1, MT2 E2
	VideoAnony	FBK	Video anonymisation	1	Fog	MT F1
	EdgeSec VPN	FORTH	Secure communications between services on infrastructure nodes	2	Fog, Cloud	MT F2, PSNC HPC
Data management and distribution subsystem	DatAna	ATOS	Collection and transformation of raw inference results – MQTT brokers and NiFi nodes	2	Fog, Cloud	MT F2, PSNC HPC
	DFB	ITML	Collection, persistent storage and fusion of SDM-compliant inference results – Kafka brokers, ES database	1	Cloud	PSNC HPC
	StreamHandler	INTRA	Persistent storage of AV data	1	Fog	MT F2
	HDD	CNR	Kafka broker partition optimisation	1	Cloud	PSNC HPC
Audio, visual and multimodal AI Subsystem	AAC	TAU	Automatic Audio Captioning	1	Fog	MT F2
	AT	TAU	Audio Tagging in fixed length segments	2	Cloud	PSNC HPC
	SED	TAU	Sound Event Detection in general audio signal	2	Cloud	PSNC HPC

	AVAD	AU	Audio-Visual Anomaly detection. Detecting deviations from normality within video frames and corresponding scene audio	2	Cloud	PSNC HPC
Optimised E2F2C processing and deployment subsystem	MARVdash	FORTH	Service deployment and Kubernetes cluster management	1	Cloud	PSNC HPC
	GPURegex	FORTH	Keyword detection using GPU acceleration	1	Fog	MT F2
E2F2C infrastructure	HPC Infrastructure	PSNC	Cloud layer for GRN4 hosting VMs for service deployment	1	Cloud	PSNC HPC
	Management and orchestration of HPC resources	PSNC	Management and monitoring via SLURM software and OpenStack.	1	Cloud	PSNC HPC
User interactions and decision-making toolkit	SmartViz	ZELUS	UI and visualisations	1	Cloud	PSNC HPC

5.4.7 MT3 – Monitoring of parking places: AI Inference runtime and deployment view

Figure 64 provides the deployment and runtime view of the MARVEL architecture for the MT3 use case.

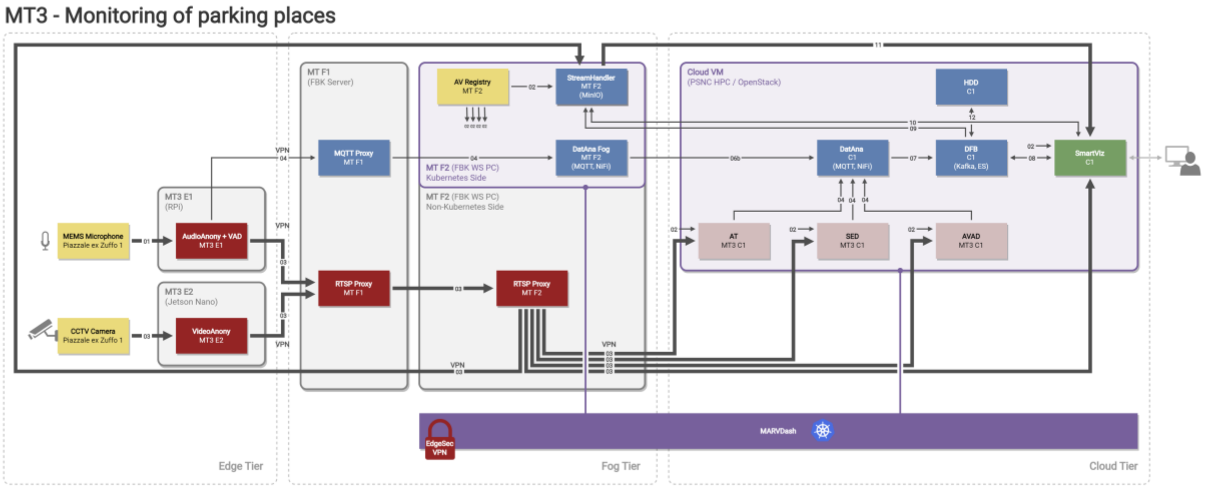


Figure 64: MARVEL R2 deployment and runtime view of the MARVEL architecture for MT3: Monitoring of parking places (*annotation label descriptions in Table 6, Section 5.2)*

This section describes the deployment, operation and interactions between components that are implemented for MT3.

An audio stream is obtained from a **MEMS microphone** that is hosted on a Raspberry Pi device connected to the network of the Municipality of Trento and located at the parking place of the Piazzale ex Zuffo at Trento. A video stream is obtained from a **CCTV camera** at the same location.

Due to restrictions associated with data privacy regulations and network security concerns expressed by the Municipality of Trento (MT), a customised solution was implemented to allow MARVEL components to access the AV data streams of the **MT3 MEMS microphone** and **CCTV camera** (similar to the solution implemented for MT1, but additionally addressing the needs of anonymising audio and video data at the edge within the MT network). According to data privacy regulations, access to the raw AV data feeds (microphone and CCTV camera) that are part of the MT network is restricted only to authorised personnel and to trusted devices within the network. Therefore, it was not possible to provide MARVEL components with direct access to the raw AV data, but only to anonymised versions of it [61]. Following deliberations with the MT authorities and based on an agreement that nominates FBK as a data processor under certain constraints, a solution was reached that involved the use of an external server (**MT F1**), which is managed by FBK. However, this server (**MT F1**) could not be attached to the MARVEL Kubernetes cluster, because such an action would violate the conditions and security policies foreseen by the agreement between MT and FBK.

It was decided to deploy **AudioAnony** (coupled with **VAD**) directly on the **MT3 E1** Raspberry Pi device and to deploy **VideoAnony** directly on the **MT3 E2** Jetson device in order to provide anonymisation at the Edge, considering that the RPi and Nvidia Jetson could provide sufficient computational power for these services. However, since **MT3 E1** and **MT3 E2** were part of the MT network, they could not be attached as a node to the MARVEL Kubernetes cluster. Furthermore, due to the agreement between MT and FBK, **MT3 E1** and **MT3 E2** could only be accessed from **MT F1** using VPN. In order to grant access to the anonymised audio from **AudioAnony** on **MT3 E1** and to the anonymised video from **VideoAnony** on **MT3 E2** to other endpoints within the MARVEL Kubernetes cluster, an **RTSP Proxy F1** service was deployed directly on **MT F1** that could re-stream the output audio stream from **AudioAnony** on **MT3 E1** and output video stream from **VideoAnony** on **MT3 E2**.

In addition, a second Fog server was set up at FBK (**MT F2**), which was allowed to (i) be attached to the MARVEL Kubernetes cluster and (ii) gain access to the **RTSP Proxy F1** and **VideoAnony** services hosted on the first infrastructure node of FBK (**MT F1**) through an **RTSP Proxy F2** service that was not part of the Kubernetes cluster, but could re-stream the output of **AudioAnony** and **VideoAnony** towards the Kubernetes cluster using a VPN tunnel. Therefore, using this configuration, it was possible for all components that are hosted within the MARVEL Kubernetes cluster to gain access to anonymised AV data streams produced by **AudioAnony** at **MT3 E1** and **VideoAnony** at **MT3 E2**.

AudioAnony on **MT3 E1** is coupled with **VAD** within the same container. **VAD** accesses the raw audio stream and detects voice activity boundaries, which are directly provided to **AudioAnony** for controlling the activation of the anonymisation process. The **VAD** component within the **AudioAnony** container on **MT3 E1** also needs to publish the voice activity boundaries it detects as raw inference results in real time to a dedicated topic of the **DatAna MQTT Proxy** broker residing at **MT F2** (within Kubernetes). For this purpose, an **MQTT Proxy** service was deployed on **MT F1** to which **VAD** could publish MQTT messages via VPN, which were then restreamed through another VPN tunnel towards the **DatAna MQTT broker on MT F2** within the Kubernetes cluster.

[61] This issue could be overcome by processing the raw AV data streams with AudioAnony and VideoAnony before feeding them to other MARVEL components. However, there were additional issues related to network security, since (i) no device that belonged in the MT network could be attached to the MARVEL Kubernetes cluster and (ii) only trusted external endpoints with compatible network security policies can be granted access to the MT network, where the Raspberry Pi (MT E1) with the MEMS microphone and the CCTV cameras are hosted.

During the system initialisation phase, the component instances that need to consume the produced anonymised AV data streams (i.e., **AT**, **SED**, **AVAD**, **StreamHandler** and **SmartViz**) request the metadata details of the **RTSP Proxy F2** AV sources (AudioAnony and VideoAnony) from the **AV Registry** (**MT F2**) via corresponding REST API calls.

In parallel, there are single instances of **AT**, **SED** and **AVAD** deployed at the **PSNC HPC**. All these components (**AT, SED, AVAD**) receive the audio RTSP stream from **AudioAnony** at **MT3 E1** in real time through the **RTSP Proxy** on **MT F2**. **AVAD** also receives the RTSP AV stream in real time from the **VideoAnony** instance on **MT3 E2** through the **RTSP Proxy** on **MT F2. AT** and **SED** process the incoming audio data stream to produce raw inference results that refer to the activity of characteristic sounds inside audio segments and to detected sound events respectively. **AVAD** processes the incoming audio and video data streams to produce raw inference results that refer to anomaly detection. All **AT**, **SED** and **AVAD** instances publish their raw inference results in real time to dedicated topics of the **DatAna MQTT broker** residing at the **PSNC HPC**.

DatAna NiFi nodes collect the inference results from corresponding **DatAna MQTT brokers** by subscribing to the respective MQTT topics. Specifically, (i) the **DatAna NiFi Fog node** on **MT F2** receives the inference results that are published on the **DatAna MQTT Fog broker** on **MT F2** and (ii) the **DatAna NiFi Cloud node** on the **PSNC HPC** receives the inference results that are published on the **DatAna MQTT Cloud broker** on the **PSNC HPC**. Each **DatAna NiFi node** performs certain operations on the collected raw inference results from all AI components to transform them to SDM-compliant data models (for more details, see Section 5.3.2). Then, each **DatAna NiFi node** pushes the transformed inference results to the **DatAna NiFi node** on the higher E/F/C layer, i.e., the **DatAna NiFi Fog node** pushes results to the **DatAna NiFi Cloud node**.

The **DatAna NiFi Cloud node** publishes the transformed inference results it collects from all layers to dedicated Kafka topics at the **DFB** (a separate topic is used for the inference results of each AI component that produced them), which is deployed at the **PSNC HPC**. The **DFB** persistently stores the received results on the **DFB ES** repository. Furthermore, the **DFB** fuses similar consecutive inference results from **AVAD, SED** and **AT** that are very close in time to generate merged events that refer to longer periods.

SmartViz is deployed at the **PSNC HPC** and accesses the incoming inference results at the **DFB** in real time by subscribing to the Kafka topics where they are published by the **DatAna NiFi Cloud node**. **SmartViz** also accesses the historical inference result data stored in the **DFB ES** database by making queries that are supported by the exposed REST API of the **DFB ES-proxy** service. **SmartViz** displays the inference results to the user through appropriate visualisations, as detailed in Section 5.5. **SmartViz** also allows the user to verify the displayed inference results and this verification information is fed back to the **DFB** by being published to a dedicated Kafka topic. The **DFB** then updates the relevant inference result entry in the **DFB ES** accordingly.

In parallel, a single instance of **StreamHandler** deployed at the Fog (**MT F2**) receives anonymised AV data streams in real time from the **RTSP Proxy** service on **MT F2**. **StreamHandler** persistently stores the incoming anonymised AV data in a MinIO repository.

The **StreamHandler** instance also accesses the incoming inference results at the **DFB** in real time by subscribing to the Kafka topics where they are published by the **DatAna NiFi Cloud node. StreamHandler** uses the information in these results (e.g., AV source and time period of result) to generate AV files that correspond to each inference result.

Upon user demand, **SmartViz** requests data from **StreamHandler** that refer to a particular inference result by accessing a REST API exposed by **StreamHandler**. **StreamHandler** composes the AV data that is requested (if not already available) and returns the location of the corresponding AV data to **SmartViz**. **SmartViz** accesses the AV data and makes it available to the user for playback within its UI.

Furthermore, upon user demand, **SmartViz** can connect to the **RTSP Proxy F2** and receive a live anonymised AV data stream via RTSP from **AudioAnony** on **MT3 E1** or from **VideoAnony** on **MT3 E2** and display it within its UI to the user. The connection is not continuous and is initiated only when a user requests a live AV feed from the location where an event has previously been detected.

The **HDD** is deployed at the **PSNC HPC** at the Cloud and can receive the current Kafka topic partition configuration from the **DFB** on a periodic basis. After analysing it, the **HDD** returns a recommendation for an optimised Kafka topic partition configuration to the **DFB**.

Finally, **MARVdash** is orchestrating the deployment of all the implemented components across the infrastructure nodes of **MT F2** and **PSNC HPC**.

Table 14 provides the list of components together with their main functionalities, across MARVEL functional subsystems that were used in MT3.

Table 14: MARVEL architectural components for MT3: Monitoring of parking places

MARVEL subsystem	Component information					
	Name	Owner	Functionality	Deployment		
				Instances	Layer	Infr. Node
Sensing and perception subsystem	MEMS microphone	IFAG	Audio data acquisition via IM69D130 and Edge processing of the acquired data	1	Edge	MT3 E1
	AV Registry	ITML	Provides information on available AV sources	1	Fog	MT F2
Security, Privacy and data protection Subsystem	AudioAnony	FBK	Audio anonymisation	1	Edge	MT3 E1
	VAD	AUD	Voice Activity Detection detects voiced segments either in batched or online fashion.	1	Edge	MT3 E1
	VideoAnony	FBK	Video anonymisation	1	Edge	MT3 E2
	EdgeSec VPN	FORTH	Secure communications between services on infrastructure nodes	2	Fog, Cloud	MT F2, PSNC HPC
Data management and distribution subsystem	DatAna	ATOS	Collection and transformation of raw inference results – MQTT brokers and NiFi nodes	2	Fog, Cloud	MT F2, PSNC HPC
	DFB	ITML	Collection, persistent storage and fusion of SDM-compliant inference results – Kafka brokers, ES database	1	Cloud	PSNC HPC
	StreamHandler	INTRA	Persistent storage of AV data	1	Fog	MT F2
	HDD	CNR	Kafka broker partition optimisation	1	Cloud	PSNC HPC

Audio, visual and multimodal AI Subsystem	AT	TAU	Audio Tagging in fixed length segments	1	Cloud	PSNC HPC
	SED	TAU	Sound Event Detection in general audio signal	1	Cloud	PSNC HPC
	AVAD	AU	Audio-Visual Anomaly detection. Detecting deviations from normality within video frames and corresponding scene audio	1	Cloud	PSNC HPC
Optimised E2F2C processing and deployment subsystem	MARVdash	FORTH	Service deployment and Kubernetes cluster management	1	Cloud	PSNC HPC
E2F2C infrastructure	HPC Infrastructure	PSNC	Cloud layer for GRN4 hosting VMs for service deployment	1	Cloud	PSNC HPC
	Management and orchestration of HPC resources	PSNC	Management and monitoring via SLURM software and OpenStack.	1	Cloud	PSNC HPC
User interactions and decision-making toolkit	SmartViz	ZELUS	UI and visualisations	1	Cloud	PSNC HPC

5.4.8 MT4 – Analysis of a specific area: AI Inference runtime and deployment view

Figure 65 provides the deployment and runtime view of the MARVEL architecture for the MT4 use case.

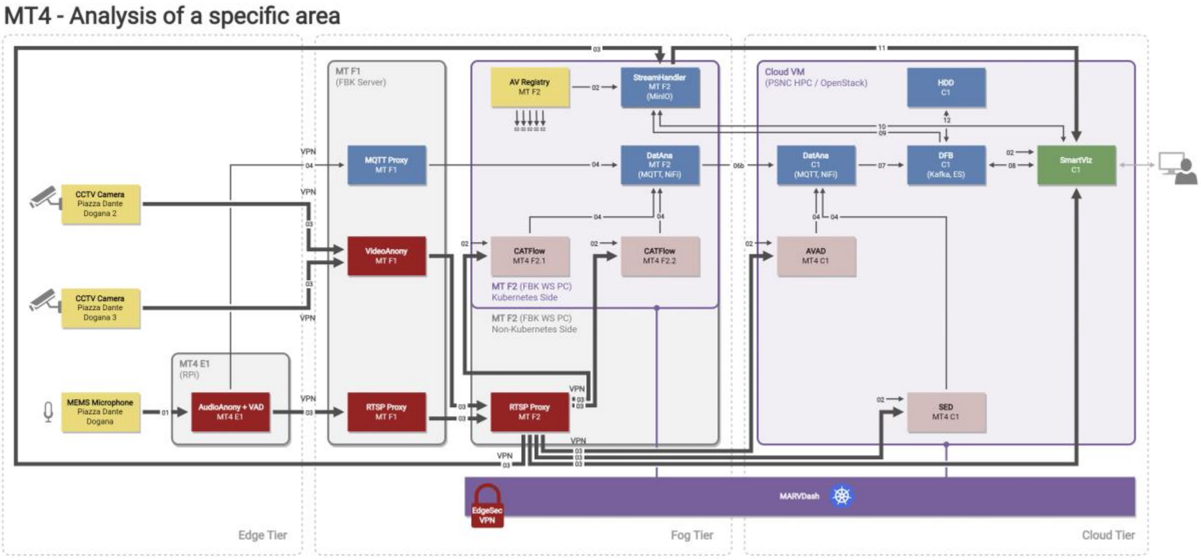


Figure 65: MARVEL R2 deployment and runtime view of the MARVEL architecture for MT4 – Analysis of a specific area (*annotation label descriptions in Table 6, Section 5.2)*

This section describes the deployment, operation and interactions between components that are implemented for MT4.

An audio stream is obtained from a **MEMS microphone** that is hosted on a Raspberry Pi device connected to the network of the Municipality of Trento and located at the Piazza Dante in Trento. Video streams are obtained from two **CCTV cameras** at the same location.

Due to restrictions associated with data privacy regulations and network security concerns expressed by the Municipality of Trento (MT), a customised solution was implemented to allow MARVEL components to access the AV data streams of the **MT4 MEMS microphone** and **CCTV cameras** (similar to the solution implemented for MT1, but additionally addressing the needs of anonymising audio at the edge within the MT network). According to data privacy regulations, access to the raw AV data feeds (microphone and CCTV cameras) that are part of the MT network is restricted only to authorised personnel and to trusted devices within the network. Therefore, it was not possible to provide MARVEL components with direct access to the raw AV data, but only to anonymised versions of it [62]. Following deliberations with the MT authorities and based on an agreement that nominates FBK as a data processor under certain constraints, a solution was reached that involved the use of an external server (**MT F1**), which is managed by FBK. However, this server (**MT F1**) could not be attached to the MARVEL Kubernetes cluster, because such an action would violate the conditions and security policies foreseen by the agreement between MT and FBK.

Regarding audio, it was decided to deploy **AudioAnony** (coupled with **VAD**) directly on the **MT4 E1 Raspberry Pi device** in order to provide anonymisation at the Edge, considering that the RPi could provide sufficient computational power for this service. However, since **MT4 E1** was part of the MT network, it could not be attached as a node to the MARVEL Kubernetes cluster. Furthermore, due to the agreement between MT and FBK, **MT4 E1** could only be accessed from **MT F1** using VPN. In order to grant access to the anonymised audio from **AudioAnony** on **MT4 E1** to other endpoints within the MARVEL Kubernetes cluster, an **RTSP Proxy F1** service was deployed directly on **MT F1** that could re-stream the output audio stream from **AudioAnony** on **MT4 E1**.

Regarding video, it was necessary to deploy a **VideoAnony** service directly on the **MT F1** server at the Fog, without it being hosted on the MARVEL Kubernetes cluster. This **VideoAnony** service could consume the raw AV streams from the CCTV cameras through a VPN tunnel.

In addition, a second Fog server was set up at FBK (**MT F2**), which was allowed to (i) be attached to the MARVEL Kubernetes cluster and (ii) gain access to the **RTSP Proxy F1** and **VideoAnony** services hosted on the first infrastructure node of FBK (**MT F1**) through an **RTSP Proxy F2** service that was not part of the Kubernetes cluster, but could re-stream the output of **AudioAnony** and **VideoAnony** towards the Kubernetes cluster using a VPN tunnel. Therefore, using this configuration, it was possible for all components that are hosted within the MARVEL Kubernetes cluster to gain access to anonymised AV data streams produced by **AudioAnony** on **MT4 E1** and by **VideoAnony** on **MT F1**.

AudioAnony on **MT4 E1** is coupled with **VAD** within the same container. **VAD** accesses the raw audio stream and detects voice activity boundaries, which are directly provided to **AudioAnony** for controlling the activation of the anonymisation process. The **VAD** component within the **AudioAnony** container on **MT4 E1** also needs to publish the voice activity

[62] This issue could be overcome by processing the raw AV data streams with AudioAnony and VideoAnony before feeding them to other MARVEL components. However, there were additional issues related to network security, since (i) no device that belonged in the MT network could be attached to the MARVEL Kubernetes cluster and (ii) only trusted external endpoints with compatible network security policies can be granted access to the MT network, where the Raspberry Pi (MT E1) with the MEMS microphone and the CCTV cameras are hosted.

boundaries it detects as raw inference results in real time to a dedicated topic of the **DatAna MQTT Proxy** broker residing at **MT F2** (within Kubernetes). For this purpose, an **MQTT Proxy** service was deployed on **MT F1** to which **VAD** could publish MQTT messages via VPN, which were then restreamed through another VPN tunnel towards the **DatAna MQTT broker on MT F2** within the Kubernetes cluster.

During system initialisation, the component instances that need to consume the produced anonymised AV data streams (i.e., **CATFlow, AVAD, SED**, **StreamHandler** and **SmartViz**) request the metadata details of the **RTSP Proxy F2** AV sources (AudioAnony and VideoAnony) from the **AV Registry** (**MT F2**) via corresponding REST API calls.

Two **CATFlow** instances are deployed at the Fog layer on **MT F2** and receive the output of anonymised AV data stream in real time from the **VideoAnony** instance on **MT F1** through the **RTSP Proxy** service on **MT F2** via RTSP to analyse the stream's video section and produce the number of detected vehicles and traffic trajectories as inference results.

Both **CATFlow** instances publish their inference results in real time to a dedicated topic of the **DatAna MQTT broker** residing at **MT F2**.

In parallel, there is one instance of **AVAD** and one instance of **SED** deployed at the **PSNC HPC.** These components (**AVAD, SED**) receive the audio RTSP stream from **AudioAnony** on **MT4 E1** in real time through the **RTSP Proxy** on **MT F2**. **AVAD** also receives the RTSP AV stream in real time from the **VideoAnony** instance on **MT F1** through the **RTSP Proxy** on **MT F2. SED** processes the incoming audio data stream to produce raw inference results that refer to sound event detection, while **AVAD** processes the incoming audio and video data streams to produce raw inference results that refer to anomaly detection. All **SED** and **AVAD** instances publish their raw inference results in real time to dedicated topics of the **DatAna MQTT broker** residing at the **PSNC HPC**.

DatAna NiFi nodes collect the inference results from corresponding **DatAna MQTT brokers** by subscribing to the respective MQTT topics. Specifically, (i) the **DatAna NiFi Fog node** on **MT F2** receives the inference results that are published on the **DatAna MQTT Fog broker** on **MT F2** and (ii) the **DatAna NiFi Cloud node** on the **PSNC HPC** receives the inference results that are published on the **DatAna MQTT Cloud broker** on the **PSNC HPC**. Each **DatAna NiFi node** performs certain operations on the collected raw inference results from all AI components to transform them to SDM-compliant data models (for more details, see Section 5.3.2). Then, each **DatAna NiFi node** pushes the transformed inference results to the **DatAna NiFi node** on the higher E/F/C layer, i.e., the **DatAna NiFi Fog node** pushes results to the **DatAna NiFi Cloud node**.

The **DatAna NiFi Cloud node** publishes the transformed inference results it collects from all layers to dedicated Kafka topics at the **DFB** (a separate topic is used for the inference results of each AI component that produced them), which is deployed at the **PSNC HPC**. The **DFB** persistently stores the received results on the **DFB ES** repository. Furthermore, the **DFB** fuses similar consecutive inference results from **AVAD** and **SED** that are very close in time to generate merged events that refer to longer periods.

SmartViz is deployed at the **PSNC HPC** and accesses the incoming inference results at the **DFB** in real time by subscribing to the Kafka topics where they are published by the **DatAna NiFi Cloud node**. **SmartViz** also accesses the historical inference result data stored in the **DFB ES** database by making queries that are supported by the exposed REST API of the **DFB ES-proxy** service. **SmartViz** displays the inference results to the user through appropriate visualisations, as detailed in Section 5.5. **SmartViz** also allows the user to verify the displayed inference results and this verification information is fed back to the **DFB** by being published to a dedicated Kafka topic. The **DFB** then updates the relevant inference result entry in the **DFB ES** accordingly.

In parallel, a single instance of **StreamHandler** deployed at the Fog (**MT F2**) receives anonymised AV data streams in real time from the **RTSP Proxy** service on **MT F2**. **StreamHandler** persistently stores the incoming anonymised AV data in a MinIO repository.

The **StreamHandler** instance also accesses the incoming inference results at the **DFB** in real time by subscribing to the Kafka topics where they are published by the **DatAna NiFi Cloud node. StreamHandler** uses the information in these results (e.g., AV source and time period of result) to generate AV files that correspond to each inference result.

Upon user demand, **SmartViz** requests data from **StreamHandler** that refer to a particular inference result by accessing a REST API exposed by **StreamHandler**. **StreamHandler** composes the AV data that is requested (if not already available) and returns the location of the corresponding AV data to **SmartViz**. **SmartViz** accesses the AV data and makes it available to the user for playback within its UI.

Furthermore, upon user demand, **SmartViz** can connect to the **RTSP Proxy F2** and receive a live anonymised AV data stream via RTSP from **AudioAnony** on **MT4 E1** or from **VideoAnony** on **MT F1** and display it within its UI to the user. The connection is not continuous and is initiated only when a user requests a live AV feed from the location where an event has previously been detected.

The **HDD** is deployed at the **PSNC HPC** at the Cloud and can receive the current Kafka topic partition configuration from the **DFB** on a periodic basis. After analysing it, the **HDD** returns a recommendation for an optimised Kafka topic partition configuration to the **DFB**.

Finally, **MARVdash** is orchestrating the deployment of all the implemented components across the infrastructure nodes of **MT F2** and **PSNC HPC**.

Table 15 provides the list of components together with their main functionalities, across MARVEL functional subsystems that were used in MT4.

Table 15: MARVEL architectural components for MT4 – Analysis of a specific area

MARVEL subsystem	Component information					
	Name	Owner	Functionality	Deployment		
				Instances	Layer	Infr. Node
Sensing and perception subsystem	MEMS microphone	IFAG	Audio data acquisition via IM69D130 and Edge processing of the acquired data	1	Edge	MT4 E1
	AV Registry	ITML	Provides information on available AV sources	1	Fog	MT F2
Security, Privacy and data protection Subsystem	AudioAnony	FBK	Audio anonymisation	1	Edge	MT4 E1
	VAD	AUD	Voice Activity Detection detects voiced segments either in batched or online fashion.	1	Edge	MT4 E1
	VideoAnony	FBK	Video anonymisation	1	Fog	MT F1

	EdgeSec VPN	FORTH	Secure communications between services on infrastructure nodes	2	Fog, Cloud	MT F2, PSNC HPC
Data management and distribution subsystem	DatAna	ATOS	Collection and transformation of raw inference results – MQTT brokers and NiFi nodes	2	Fog, Cloud	MT F2, PSNC HPC
	DFB	ITML	Collection, persistent storage and fusion of SDM-compliant inference results – Kafka brokers, ES database	1	Cloud	PSNC HPC
	StreamHandler	INTRA	Persistent storage of AV data	1	Fog	MT F2
	HDD	CNR	Kafka broker partition optimisation	1	Cloud	PSNC HPC
Audio, visual and multimodal AI Subsystem	CATFlow	GRN	AI inference: pedestrian/vehicle count, traffic trajectories	2	Fog	MT F2
	SED	TAU	Sound Event Detection in general audio signal	1	Cloud	PSNC HPC
	AVAD	AU	Audio-Visual Anomaly detection. Detecting deviations from normality within video frames and corresponding scene audio	1	Cloud	PSNC HPC
Optimised E2F2C processing and deployment subsystem	MARVdash	FORTH	Service deployment and Kubernetes cluster management	1	Cloud	PSNC HPC
E2F2C infrastructure	HPC Infrastructure	PSNC	Cloud layer for GRN4 hosting VMs for service deployment	1	Cloud	PSNC HPC
	Management and orchestration of HPC resources	PSNC	Management and monitoring via SLURM software and OpenStack.	1	Cloud	PSNC HPC
User interactions and decision-making toolkit	SmartViz	ZELUS	UI and visualisations	1	Cloud	PSNC HPC

5.4.9 UNS1 – Drone Experiment: AI Inference runtime and deployment view

Figure 66 provides the deployment and runtime view of the MARVEL architecture for the UNS1 use case.

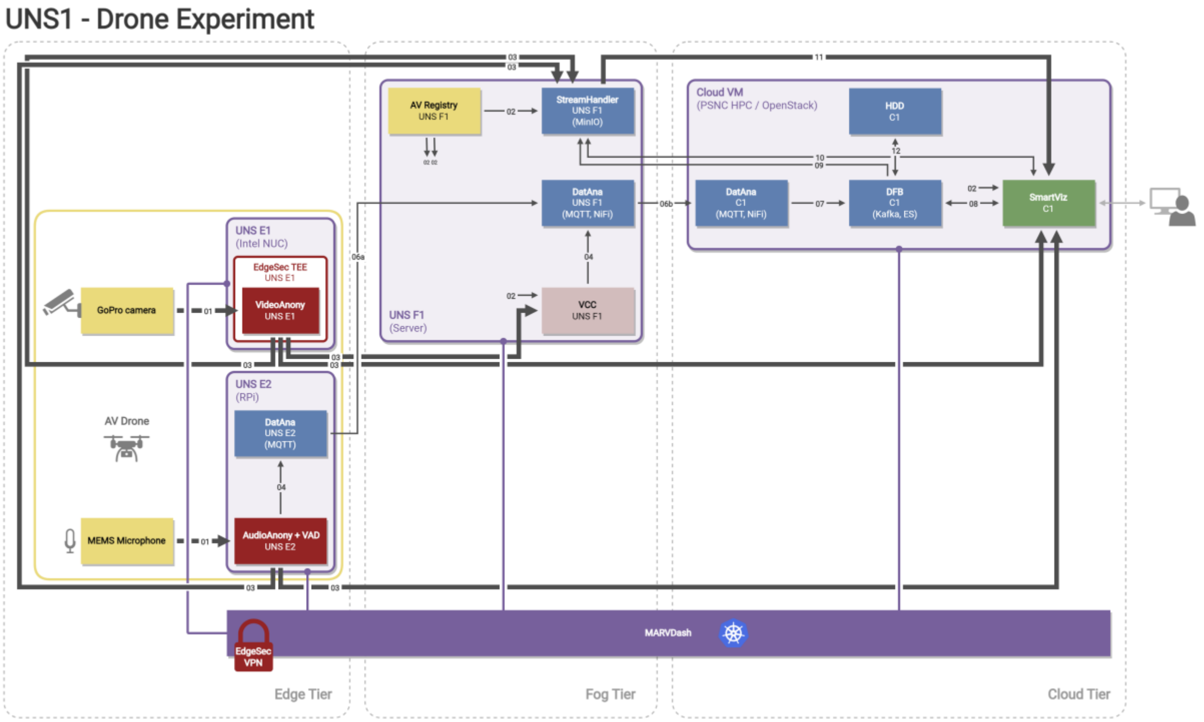


Figure 66: MARVEL R2 deployment and runtime view of the MARVEL architecture for UNS1: Drone Experiment (*annotation label descriptions in Table 6, Section 5.2)*

This section describes the deployment, operation and interactions between components that are implemented for UNS1.

An audio stream is obtained from a **MEMS microphone** that is hosted on a Raspberry Pi device (**UNS E2**), located in Novi Sad. A video stream is obtained from a **GoPro camera, which is attached to an Intel NUC device** (**UNS E1**). The MEMS microphone, the Raspberry Pi, the GoPro camera and the Intel NUC are mounted on the **AVDrone**, which is a hexacopter UAV, controlled by a human operator.

One **AudioAnony** instance is deployed on **UNS E2** that processes the AV data stream from the **MEMS** microphone hosted on the same device. One **VideoAnony** instance is deployed within **EdgeSec TEE** on **UNS E1. EdgeSec TEE** ensures the trustworthiness of the **VideoAnony** instance it encapsulates and protects the access to the live video stream of the GoPro camera that **VideoAnony** requires.

AudioAnony on **UNS E2** is coupled with **VAD** within the same container. **VAD** accesses the raw audio stream and detects voice activity boundaries, which are directly provided to **AudioAnony** for controlling the activation of the anonymisation process.

During system initialisation, the component instances that need to consume the produced anonymised AV data streams (i.e., **VCC**, **StreamHandler** and **SmartViz**) request the metadata details of the **AudioAnony** and **VideoAnony** AV sources from the **AV Registry**, which is deployed at a server at the premises of the University of Novi Sad (**UNS F1)** via corresponding REST API calls.

The **VAD** component within the **AudioAnony** container on **UNS E2** publishes the voice activity boundaries it detects as raw inference results in real time to a dedicated topic of the **DatAna MQTT broker** at **UNS E1**.

In parallel, there is a single instance of **VCC** deployed at the **UNS F1**, which receives the RTSP AV stream in real time from the **VideoAnony** instance on **UNS E1. VCC** processes the incoming video data stream to produce raw inference results that refer to crowd counting and the generation of visual heatmaps. The **VCC** service publishes its raw inference results in real time to a dedicated topic of the **DatAna MQTT broker** residing at the **UNS F1**.

DatAna NiFi nodes collect the inference results from **DatAna MQTT brokers** by subscribing to the respective MQTT topics. Specifically, (i) the **DatAna NiFi Fog node** on **UNS F1** receives the inference results that are published on the **DatAna MQTT Edge broker** on **UNS E2** and the **DatAna MQTT Fog broker** on **UNS F1** and (ii) the **DatAna NiFi Cloud node** on the **PSNC HPC** receives the inference results that are published on the **DatAna MQTT Cloud broker** on the **PSNC HPC**. Each **DatAna NiFi node** performs certain operations on the collected raw inference results from all AI components to transform them to SDM-compliant data models (for more details, see Section 5.3.2. Then, each **DatAna NiFi node** pushes the transformed inference results to the **DatAna NiFi node** on the higher E/F/C layer, i.e., (i) the **DatAna NiFi Edge nodes** push results to the respective **DatAna NiFi Fog node** and (ii) the **DatAna NiFi Fog nodes** push results to the **DatAna NiFi Cloud node**.

The **DatAna NiFi Cloud node** publishes the transformed inference results it collects from all layers to dedicated Kafka topics at the **DFB** (a separate topic is used for the inference results of each AI component that produced them), which is deployed at the **PSNC HPC**. The **DFB** persistently stores the received results on the **DFB ES** repository.

SmartViz is deployed at the **PSNC HPC** and accesses the incoming inference results at the **DFB** in real time by subscribing to the Kafka topics where they are published by the **DatAna NiFi Cloud node**. **SmartViz** also accesses the historical inference result data stored in the **DFB ES** database by making queries that are supported by the exposed REST API of the **DFB ES-proxy** service. **SmartViz** displays the inference results to the user through appropriate visualisations, as detailed in Section 5.5. **SmartViz** also allows the user to verify the displayed inference results and this verification information is fed back to the **DFB** by being published to a dedicated Kafka topic. The **DFB** then updates the relevant inference result entry in the **DFB ES** accordingly.

In parallel, a single instance of **StreamHandler** deployed at the Fog (**UNS F1**) receives anonymised AV data streams in real time from the **AudioAnony** service at **UNS E2** and the **VideoAnony** service at **UNS E1**. **StreamHandler** persistently stores the incoming anonymised AV data in a MinIO repository.

The **StreamHandler** instance also accesses the incoming inference results at the **DFB** in real time by subscribing to the Kafka topics where they are published by the **DatAna NiFi Cloud node. StreamHandler** uses the information in these results (e.g., AV source and time period of result) to generate AV files that correspond to each inference result.

Upon user demand, **SmartViz** requests data from **StreamHandler** that refer to a particular inference result by accessing a REST API exposed by **StreamHandler**. **StreamHandler** composes the AV data that is requested (if not already available) and returns the location of the corresponding AV data to **SmartViz**. **SmartViz** accesses the AV data and makes it available to the user for playback within its UI.

Furthermore, upon user demand, **SmartViz** can connect to the **AudioAnony** service at **UNS E2** or the **VideoAnony** service at **UNS E1** to receive a live AV data stream via RTSP and display it within its UI to the user. The connection is not continuous and is initiated only when a user requests a live AV feed from the location where an event has previously been detected.

The **HDD** is deployed at the **PSNC HPC** at the Cloud and can receive the current Kafka topic partition configuration from the **DFB** on a periodic basis. After analysing it, the **HDD** returns a recommendation for an optimised Kafka topic partition configuration to the **DFB**.

Finally, **MARVdash** is orchestrating the deployment of all the implemented components across the infrastructure nodes of **UNS E1**, **UNS E2**, **UNS F1** and **PSNC HPC**.

Table 16 provides the list of components together with their main functionalities, across MARVEL functional subsystems that were used in UNS1.

Table 16: MARVEL architectural components for UNS1: Drone Experiment

MARVEL subsystem	Component information					
	Name	Owner	Functionality	Deployment		
				Instances	Layer	Infr. Node
Sensing and perception subsystem	MEMS microphone	IFAG	Audio data acquisition via IM69D130 and Edge processing of the acquired data	1	Edge	UNS E1
	AVDrone	UNS	Drone-mounted equipment for audio and video data acquisition and streaming	1	Edge	N/A
	AV Registry	ITML	Provides information on available AV sources	1	Fog	UNS F1
Security, Privacy and data protection Subsystem	AudioAnony	FBK	Audio anonymisation	1	Edge	UNS E2
	VAD	AUD	Voice Activity Detection detects voiced segments either in batched or online fashion.	1	Edge	UNS E2
	VideoAnony	FBK	Video anonymisation	1	Edge	UNS E1
	EdgeSec TEE	FORTH	Secure computing based on Intel SGX security features.	4	Edge	UNS E1
	EdgeSec VPN	FORTH	Secure communications between services on infrastructure nodes	4	Edge, Fog, Cloud	UNS E1, UNS E2, UNS F1, PSNC HPC
Data management and distribution subsystem	DatAna	ATOS	Collection and transformation of raw inference results – MQTT brokers and NiFi nodes	3	Edge, Fog, Cloud	UNS E1, UNS F1, PSNC HPC
	DFB	ITML	Collection, persistent storage and fusion of SDM-compliant inference results – Kafka brokers, ES database	1	Cloud	PSNC HPC
	StreamHandler	INTRA	Persistent storage of AV data	1	Fog	UNS F1
	HDD	CNR	Kafka broker partition optimisation	1	Cloud	PSNC HPC
Audio, visual and multimodal AI Subsystem	VCC	AU	Visual Crowd Counting. Total number of people present in given video frames.	1	Fog	UNS F1

Optimised E2F2C processing and deployment subsystem	MARVdash	FORTH	Service deployment and Kubernetes cluster management	1	Cloud	PSNC HPC
E2F2C infrastructure	HPC Infrastructure	PSNC	Cloud layer for GRN4 hosting VMs for service deployment	1	Cloud	PSNC HPC
	Management and orchestration of HPC resources	PSNC	Management and monitoring via SLURM software and OpenStack.	1	Cloud	PSNC HPC
User interactions and decision-making toolkit	SmartViz	ZELUS	UI and visualisations	1	Cloud	PSNC HPC

5.4.10 UNS2 – Localising audio events in crowds: AI Inference runtime and deployment view

Figure 67 provides the deployment and runtime view of the MARVEL architecture for the UNS1 use case.

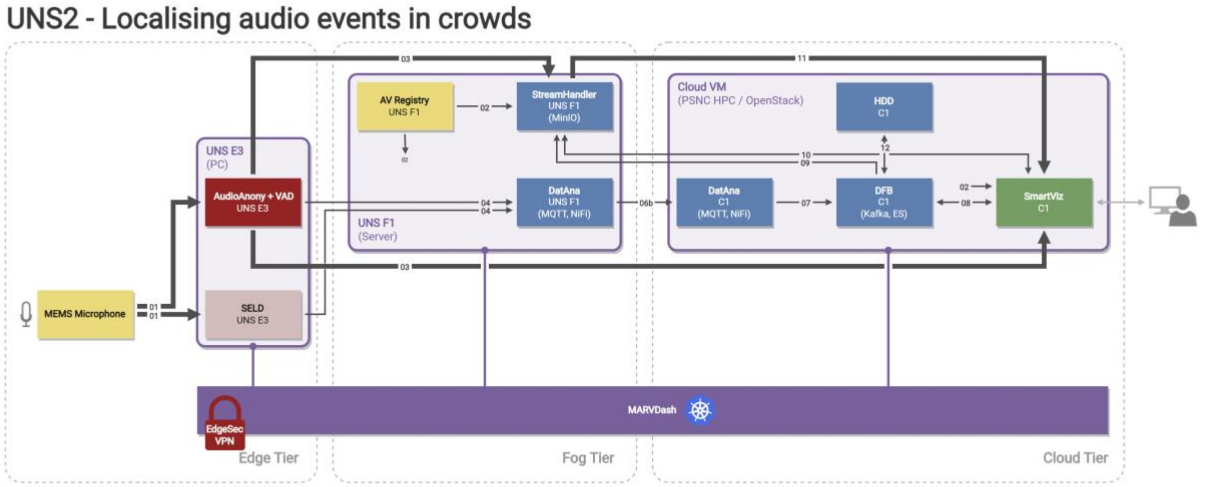


Figure 67: MARVEL R2 deployment and runtime view of the MARVEL architecture for UNS2 – Localising audio events in crowds (*annotation label descriptions in Table 6, Section 5.2)*

This section describes the deployment, operation and interactions between components that are implemented for UNS2.

An 8-channel audio stream is obtained from a **MEMS microphone** that is attached to a portable PC device (**UNS E3**), located in Novi Sad.

One **AudioAnony** instance is deployed on **UNS E3** that processes the AV data stream from the **MEMS** microphone hosted on the same device. Only one of the eight channels produced by the **MEMS microphone** is processed for anonymisation and streaming by **AudioAnony**.

AudioAnony on **UNS E2** is coupled with **VAD** within the same container. **VAD** accesses the raw audio stream and detects voice activity boundaries, which are directly provided to **AudioAnony** for controlling the activation of the anonymisation process.

One SELD instance is also deployed on UNS E2 and it processes all eight audio channels of the AV data stream from the **MEMS** microphone hosted on the same device.

During system initialisation, the component instances that need to consume the produced anonymised AV data streams (i.e., **StreamHandler** and **SmartViz**) request the metadata details of the **AudioAnony** AV sources from the **AV Registry**, which is deployed at a server at the premises of the University of Novi Sad (**UNS F1)** via corresponding REST API calls.

The **VAD** component within the **AudioAnony** container on **UNS E3** publishes the voice activity boundaries it detects as raw inference results in real time to a dedicated topic of the **DatAna MQTT broker** at **UNS F1**.

The **SELD** component on **UNS E3** also publishes its inference results in real time to a dedicated topic of the **DatAna MQTT broker** at **UNS F1**. The inference results refer to sound detection and localisation.

DatAna NiFi nodes collect the inference results from **DatAna MQTT brokers** by subscribing to the respective MQTT topics. Specifically, (i) the **DatAna NiFi Fog node** on **UNS F1** receives the inference results that are published on the **DatAna MQTT Fog broker** on **UNS F1**. Each **DatAna NiFi node** performs certain operations on the collected raw inference results from all AI components to transform them to SDM-compliant data models (for more details, see Section 5.3.2). Then, each **DatAna NiFi node** pushes the transformed inference results to the **DatAna NiFi node** on the higher E/F/C layer, i.e., (i) the **DatAna NiFi Edge nodes** push results to the respective **DatAna NiFi Fog node** and (ii) the **DatAna NiFi Fog nodes** push results to the **DatAna NiFi Cloud node**.

The **DatAna NiFi Cloud node** publishes the transformed inference results it collects from all layers to dedicated Kafka topics at the **DFB** (a separate topic is used for the inference results of each AI component that produced them), which is deployed at the **PSNC HPC**. The **DFB** persistently stores the received results on the **DFB ES** repository.

SmartViz is deployed at the **PSNC HPC** and accesses the incoming inference results at the **DFB** in real time by subscribing to the Kafka topics where they are published by the **DatAna NiFi Cloud node**. **SmartViz** also accesses the historical inference result data stored in the **DFB ES** database by making queries that are supported by the exposed REST API of the **DFB ES-proxy** service. **SmartViz** displays the inference results to the user through appropriate visualisations, as detailed in Section 5.5. **SmartViz** also allows the user to verify the displayed inference results and this verification information is fed back to the **DFB** by being published to a dedicated Kafka topic. The **DFB** then updates the relevant inference result entry in the **DFB ES** accordingly.

In parallel, a single instance of **StreamHandler** deployed at the Fog (**UNS F1**) receives anonymised AV data streams in real time from the **AudioAnony** service at **UNS E3**. **StreamHandler** persistently stores the incoming anonymised AV data in a MinIO repository.

The **StreamHandler** instance also accesses the incoming inference results at the **DFB** in real time by subscribing to the Kafka topics where they are published by the **DatAna NiFi Cloud node. StreamHandler** uses the information in these results (e.g., AV source and time period of result) to generate AV files that correspond to each inference result.

Upon user demand, **SmartViz** requests data from **StreamHandler** that refer to a particular inference result by accessing a REST API exposed by **StreamHandler**. **StreamHandler** composes the AV data that is requested (if not already available) and returns the location of the corresponding AV data to **SmartViz**. **SmartViz** accesses the AV data and makes it available to the user for playback within its UI.

Furthermore, upon user demand, **SmartViz** can connect to the **AudioAnony** service at **UNS E3** to receive a live AV data stream via RTSP and present it within its UI to the user. The connection is not continuous and is initiated only when a user requests a live AV feed from the location where an event has previously been detected.

The **HDD** is deployed at the **PSNC HPC** at the Cloud and can receive the current Kafka topic partition configuration from the **DFB** on a periodic basis. After analysing it, the **HDD** returns a recommendation for an optimised Kafka topic partition configuration to the **DFB**.

Finally, **MARVdash** is orchestrating the deployment of all the implemented components across the infrastructure nodes of **UNS E3**, **UNS F1** and **PSNC HPC**.

Table 17 provides the list of components together with their main functionalities, across MARVEL functional subsystems that were used in UNS1.

Table 17: MARVEL architectural components for UNS1: Drone Experiment

MARVEL subsystem	Component information					
	Name	Owner	Functionality	Deployment		
				Instances	Layer	Infr. Node
Sensing and perception subsystem	MEMS microphone	IFAG	Audio data acquisition via IM69D130 and Edge processing of the acquired data	1	Edge	UNS E3
	AV Registry	ITML	Provides information on available AV sources	1	Fog	UNS F1
Security, Privacy and data protection Subsystem	AudioAnony	FBK	Audio anonymisation	1	Edge	UNS E3
	VAD	AUD	Voice Activity Detection detects voiced segments either in batched or online fashion.	1	Edge	UNS E3
	EdgeSec VPN	FORTH	Secure communications between services on infrastructure nodes	3	Edge, Fog, Cloud	UNS E3, UNS F1, PSNC HPC
Data management and distribution subsystem	DatAna	ATOS	Collection and transformation of raw inference results – MQTT brokers and NiFi nodes	2	Fog, Cloud	UNS F1, PSNC HPC
	DFB	ITML	Collection, persistent storage and fusion of SDM-compliant inference results – Kafka brokers, ES database	1	Cloud	PSNC HPC
	StreamHandler	INTRA	Persistent storage of AV data	1	Fog	UNS F1
	HDD	CNR	Kafka broker partition optimisation	1	Cloud	PSNC HPC
Audio, visual and multimodal AI Subsystem	SELD	TAU	Sound Event Localisation and Detection	1	Edge	UNS E3
Optimised E2F2C processing and deployment subsystem	MARVdash	FORTH	Service deployment and Kubernetes cluster management	1	Cloud	PSNC HPC

E2F2C infrastructure	HPC Infrastructure	PSNC	Cloud layer for GRN4 hosting VMs for service deployment	1	Cloud	PSNC HPC
	Management and orchestration of HPC resources	PSNC	Management and monitoring via SLURM software and OpenStack.	1	Cloud	PSNC HPC
User interactions and decision-making toolkit	SmartViz	ZELUS	UI and visualisations	1	Cloud	PSNC HPC

5.4.11 AI Training runtime and deployment view

Figure 68 provides the deployment and runtime view of the MARVEL reference architecture for AI Training that is applied to support the 'AI Inference Pipeline' in all R2 use cases.

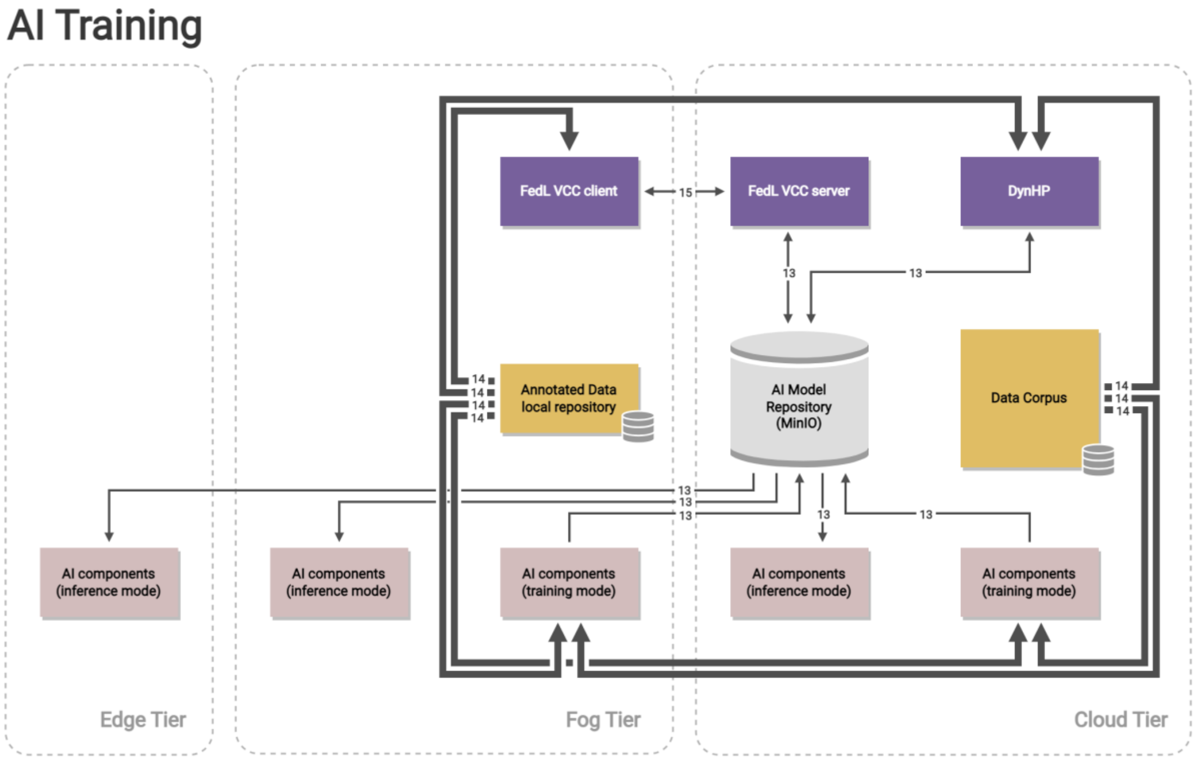

AI Training
FedL VCC client
15
FedL VCC server
DynHP
13
13
14
14
14
14
Annotated Data local repository
AI Model Repository (MinIO)
Data Corpus
14
14
14
13
13
13
13
13
AI components (inference mode)
AI components (inference mode)
AI components (training mode)
AI components (inference mode)
AI components (training mode)
Edge Tier
Fog Tier
Cloud Tier

Figure 68: MARVEL R2 deployment and runtime view of the MARVEL architecture for AI Training *(annotation label descriptions in Table 6, Section 5.2)*

MARVEL has established a methodology and an associated system architecture that are dedicated to AI Training, serving the purpose of the continuous improvement of the 'AI Inference Pipeline', which is applied to all R2 use cases.

To a large extent, the operation of the MARVEL framework in the 'AI Inference Pipeline' relies on AI components that require Machine Learning (ML), Deep Learning (DL) and Federated Learning for the formulation and optimisation of their associated AI models. While each component may adopt a different approach to perform such AI Training processes according to its nature and objectives, all these processes depend on the availability of relevant datasets and annotation information. In addition, some of the MARVEL AI models are significantly case-specific, i.e., to speed-up the process and improve the inference performance, they might require being pre-trained using datasets that are compatible with use-case specific input data.

AI Training of MARVEL AI components can occur using each component's own resources and using private or publicly available datasets that have not necessarily emerged from the MARVEL data collection activities. However, in the context of the MARVEL R2 integration activities, dedicated processes were established to support AI training of MARVEL components that took the form of a structured system architecture. The AI Training processes foreseen by this architecture can be activated on an on-demand basis or be driven by a periodical schedule, subject to the availability of new datasets in the Data Corpus. The associated AI Training architecture ensures that:

- AI components in training mode can gain access to the data that is aggregated at the Data Corpus or other local data repositories on the MARVEL infrastructure for AI training purposes.
- AI components can store and retrieve different versions of their associated AI models that are produced through AI Training with different datasets.
- AI models can be further enhanced using special compression and optimisation techniques to provide improved performance in the 'AI Inference Pipeline' (DynHP).
- Federated Learning can be applied in AI training processes to enable model training over private, distributed datasets (FedL).

This section describes the deployment, operation and interactions between components that are implemented in the MARVEL AI Training architecture.

AI components in training mode deployed at the Fog or Cloud infrastructure nodes may access the **Data Corpus** that is deployed at the Cloud **PSNC HPC** and/or other local data repositories hosted on infrastructure at the Fog layer to retrieve raw datasets and annotation information and use these data to feed their own AI training processes. The training process typically also involves a quality control phase (benchmarking), which can be automated or manual to ensure that the trained model meets certain functional and performance criteria.

After concluding AI training tasks, **AI components in training mode** can store the updated AI models that they produce in a central **AI Model Repository** that is hosted at the Cloud **PSNC HPC** using specified data structures and naming conventions (see Section 5.3.4).

AI components in inference mode deployed at the Edge, Fog or Cloud infrastructure nodes can access and retrieve any AI model that is associated with their functions from the central **AI Model Repository** that is hosted at the Cloud **PSNC HPC**.

DynHP is hosted at the Fog or Cloud layer and can access the **AI Model Repository** to retrieve an AI model that is foreseen to be used by a MARVEL AI component. **DynHP** then re-trains the model using techniques for compression and optimisation. During this process, **DynHP** may also retrieve datasets from the **Data Corpus**. The resulting optimised and compressed AI model is stored back in the **AI Model Repository**.

The **FedL** component is composed of a client and a server application. Typically, the **FedL server** is hosted at the Cloud node and the **FedL client** is hosted at a Fog node. The **FedL server** receives a compressed/optimised AI model from the **AI Model Repository**. Subsequently, the **FedL server** exchanges information bidirectionally with the **FedL client** to process a compressed/optimised AI model and update it after performing federated learning. At the end of this process, the **FedL server** returns an updated version of the model to the **AI Model Repository**.

Table 18 provides the list of components together with their main functionalities, across MARVEL functional subsystems that are used in the MARVEL AI Training architecture.

Table 18: MARVEL architectural components for AI Training

MARVEL subsystem	Component information		
	Name	**Owner**	**Functionality**
Security, Privacy and data protection Subsystem	VideoAnony	FBK	Video anonymisation
	AudioAnony	FBK	Audio anonymisation
Audio, visual and multimodal AI Subsystem	TAD	GRN	Text Anomaly Detection. Analysis of CATFlow raw inference results
	SED	TAU	Sound Event Detection in general audio signal
	AT	TAU	Audio Tagging in fixed length segments
	AAC	TAU	Automated Audio Captioning
	SELD	TAU	Sound Event Localisation and Detection
	ViAD	AU	Visual Anomaly detection. Detecting deviations from normality within video frames
	AVAD	AU	Audio-Visual Anomaly detection. Detecting deviations from normality within video frames and corresponding scene audio
	VCC	AU	Visual Crowd Counting. Total number of people present in given video frames.
	AVCC	AU	Audio-Visual Crowd Counting. Total number of people present in given video frames and from the corresponding ambient audio.
	YOLO-SED	AU, TAU	Anomaly detection using YOLO object detector and SED audio analysis
	RBAD	AU	Rule-Based Anomaly Detector
Optimised E2F2C processing and deployment subsystem	DynHP	CNR	Training and compressing a deep neural network model under a fixed memory budget for model's deployment at Edge/Fog layer.
	FedL	UNS	Algorithmic and protocol framework for federated learning (FL).
	MARVdash	FORTH	Service deployment and Kubernetes cluster management
E2F2C infrastructure	HPC Infrastructure	PSNC	Cloud layer for GRN4 hosting VMs for service deployment
	Management and orchestration of HPC resources	PSNC	Management and monitoring via SLURM software and OpenStack.
	Data Corpus	STS	A data pool of anonymised and labelled data and inference results for AI training purposes

5.4.12 Data Corpus Data Aggregation runtime and deployment view

Figure 69 provides the deployment and runtime view of the MARVEL reference architecture for the Data Corpus that is applicable in all use cases addressed by R2.

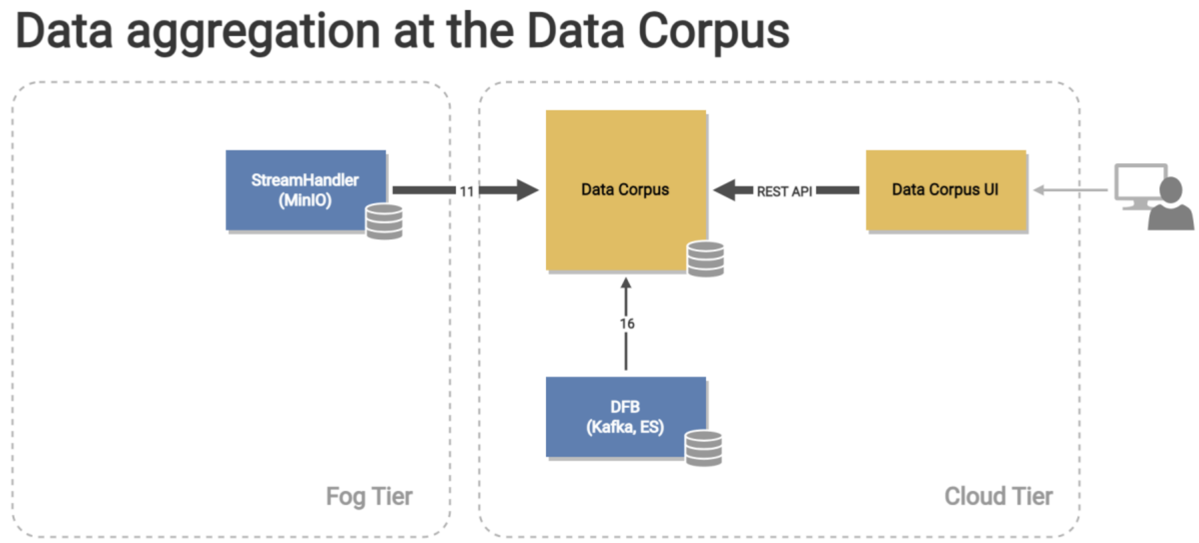


Figure 69: MARVEL R2 deployment and runtime view of the MARVEL architecture for Data Aggregation at the Data Corpus (*annotation label descriptions in Table 6, Section 5.2*

The AI Training pipeline is supported by another mechanism associated with the **Data Corpus**, whose purpose is to populate the **Data Corpus** with the necessary data that can be used for AI Training. This section describes the operation and interactions between components that are implemented for the Data Corpus Data Aggregation.

The **Data Corpus** is hosted on the **PSNC HPC** and incorporates a front-end client with a GUI that allows users to upload AV datasets that are coupled with associated annotations for AI Training. The client accesses the Data Corpus API to transmit the data to the Data Corpus back-end repository for persistent storage.

In addition, the **Data Corpus** can access the MinIO repository of each **StreamHandler** instance that is deployed at the Fog nodes of each pilot (GRN, MT, UNS) for the needs of the 'AI Inference Pipeline' of each use case. Each **StreamHandler** instance collects AV data by consuming the RTSP streams from instances of **AudioAnony** and **VideoAnony** deployed at the Edge and Fog nodes of the respective pilot, which anonymise the raw AV data streams from the capturing devices at the Edge. **StreamHandler** segments the incoming AV data streams into binary AV files and stores them in its MinIO repository. The **Data Corpus** can retrieve AV files directly from the **StreamHandler MinIO** repository.

The **Data Corpus** also maintains a copy of inference results that are being produced by the 'AI Inference Pipeline' that is applied in all R2 use cases. Since the **DFB** aggregates the inference results from all use cases, the **Data Corpus** can receive them during the operation of the 'AI Inference Pipeline' by subscribing to the **DFB** Kafka topics, where they are published by **DatAna**. The **Data Corpus** also accesses the **DFB** Kafka topic, where **SmartViz** publishes information related to the verification of inference results by human operators. This information is particularly valuable for the **Data Corpus** as it constitutes a "ground truth" and can provide labelled datasets for future AI training activities, which are fully aligned with the needs of the MARVEL AI components.

Table 19 provides the list of components together with their main functionalities, across MARVEL functional subsystems that are used for the Data Corpus Data Aggregation.

Table 19: MARVEL architectural components for Data Corpus

MARVEL subsystem	Component information		
	Name	Owner	Functionality
Data management and distribution subsystem	DFB	ITML	Collection, persistent storage and fusion of SDM-compliant inference results – Kafka brokers, ES database
	StreamHandler	INTRA	Persistent storage of AV data
Optimised E2F2C processing and deployment subsystem	MARVdash	FORTH	Service deployment and Kubernetes cluster management
E2F2C infrastructure	HPC Infrastructure	PSNC	Cloud layer for GRN4 hosting VMs for service deployment
	Management and orchestration of HPC resources	PSNC	Management and monitoring via SLURM software and OpenStack.
	Data Corpus	STS	A data pool of anonymised and labelled data and inference results for AI training purposes

5.5 UI/UX Design

The DMT serves as the UI for the MARVEL project. In collaboration with the pilots, the needs and objectives of the use cases were identified and addressed. The process followed involved analysing the user requirements, defining both functional and non-functional requirements for the DMT functionalities, and mapping them to the visualisation widgets. Finally, mockup screens that compose the user interface were developed and reviewed by end users who provided feedback that was used to design the final form of the interface. This systematic process ensures that the DMT effectively aligns with user needs and provides a comprehensive interface for decision-making within the MARVEL project.

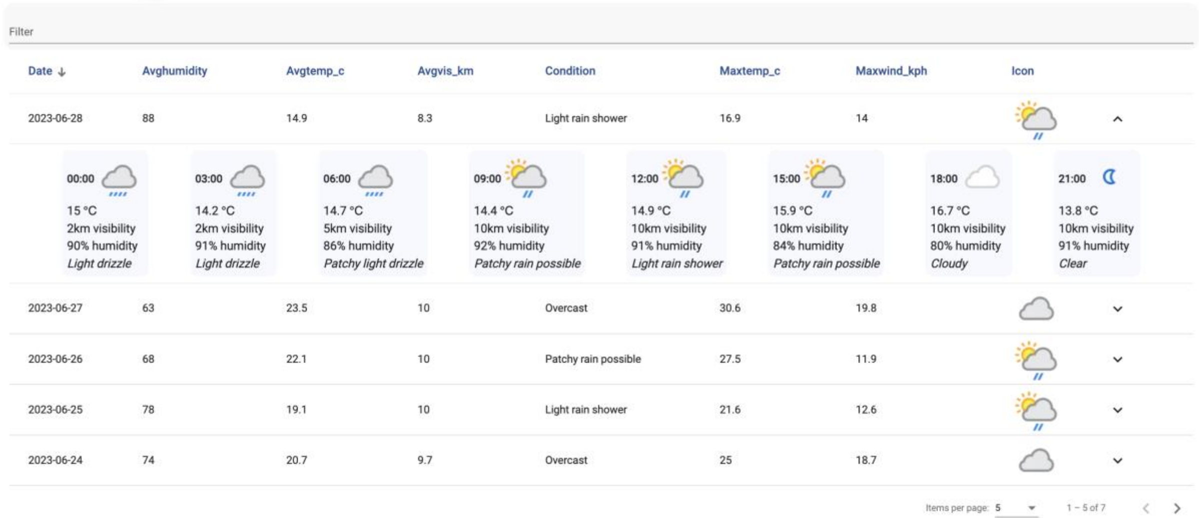
Through the aforementioned process, the DMT offers to the users a range of visualisations for detected events, anomalies, and alerts. These visualisations are accompanied by relevant audio and video snippets, bus scheduling information correlated with some cameras, weather-related data, and more. Further details on the DMT and its functionalities are included in Section 2.3 of D4.6 *'Decision Making Toolkit-final version'*.

The toolkit enables users to visualise events, alerts, and anomalies, providing valuable insights for making medium to long-term decisions. To ensure user-friendliness and intuitiveness, the DMT incorporates preconfigured visualisations that have been carefully selected based on best practices and ZELUS team's experience. This allows users to utilise the toolkit effectively without the need for extensive training, enhancing its accessibility and usability.

The visualisation widgets and functionalities that have been added in R2 are presented in the following sub-sections. D4.6 presents all the visualisation widgets (including the ones developed during R1) in more detail.

5.5.1 Weather Information

The Weather Information widget (Figure 70) provides a representation of weather-related data for the analysed area within the MARVEL use cases. Users can interact with this widget to view weather information for a selected time period, with the ability to drill down into 3-hour time windows. The widget displays details such as visibility, humidity, temperature, and an overall summary of the weather conditions in the area. By visualising these weather variables, the widget helps users understand the weather factors that may contribute to the detection process and events within the MARVEL system.



Filter

Date ↓	Avghumidity	Avgtemp_c	Avgvis_km	Condition	Maxtemp_c	Maxwind_kph	Icon
2023-06-28	88	14.9	8.3	Light rain shower	16.9	14	🌦
2023-06-27	63	23.5	10	Overcast	30.6	19.8	☁
2023-06-26	68	22.1	10	Patchy rain possible	27.5	11.9	🌦
2023-06-25	78	19.1	10	Light rain shower	21.6	12.6	🌦
2023-06-24	74	20.7	9.7	Overcast	25	18.7	☁

00:00	03:00	06:00	09:00	12:00	15:00	18:00	21:00
15 °C	14.2 °C	14.7 °C	14.4 °C	14.9 °C	15.9 °C	16.7 °C	13.8 °C
2km visibility	2km visibility	5km visibility	10km visibility	10km visibility	10km visibility	10km visibility	10km visibility
90% humidity	91% humidity	86% humidity	92% humidity	91% humidity	84% humidity	80% humidity	91% humidity
Light drizzle	*Light drizzle*	*Patchy light drizzle*	*Patchy rain possible*	*Light rain shower*	*Patchy rain possible*	*Cloudy*	*Clear*

Items per page: 5 1 – 5 of 7

Figure 70: Weather widget

5.5.2 Sound Localisation Map

The Sound Localisation Map widget (Figure 71) consists of a map showcasing the direction of a sound event detected by the SELD component. The user who interacts with this widget can view consecutive available detected events with their direction indicated by an arrow from a selected time period.

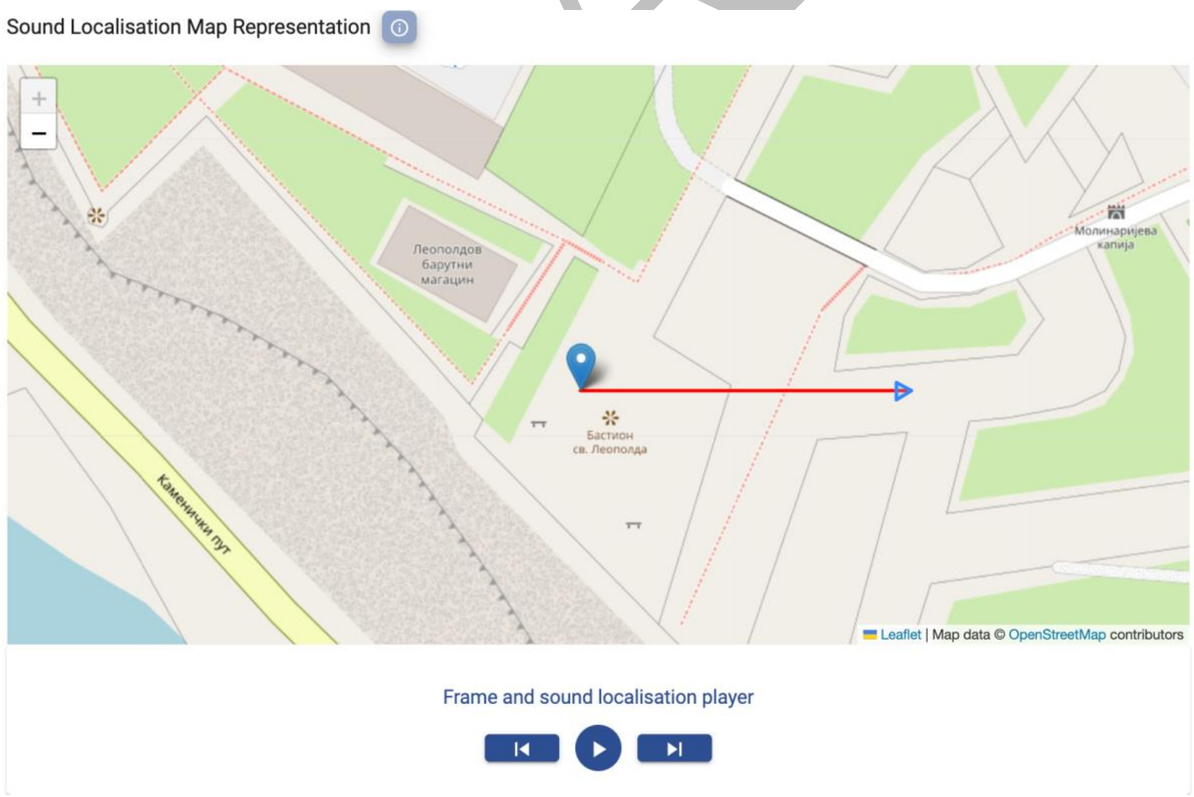
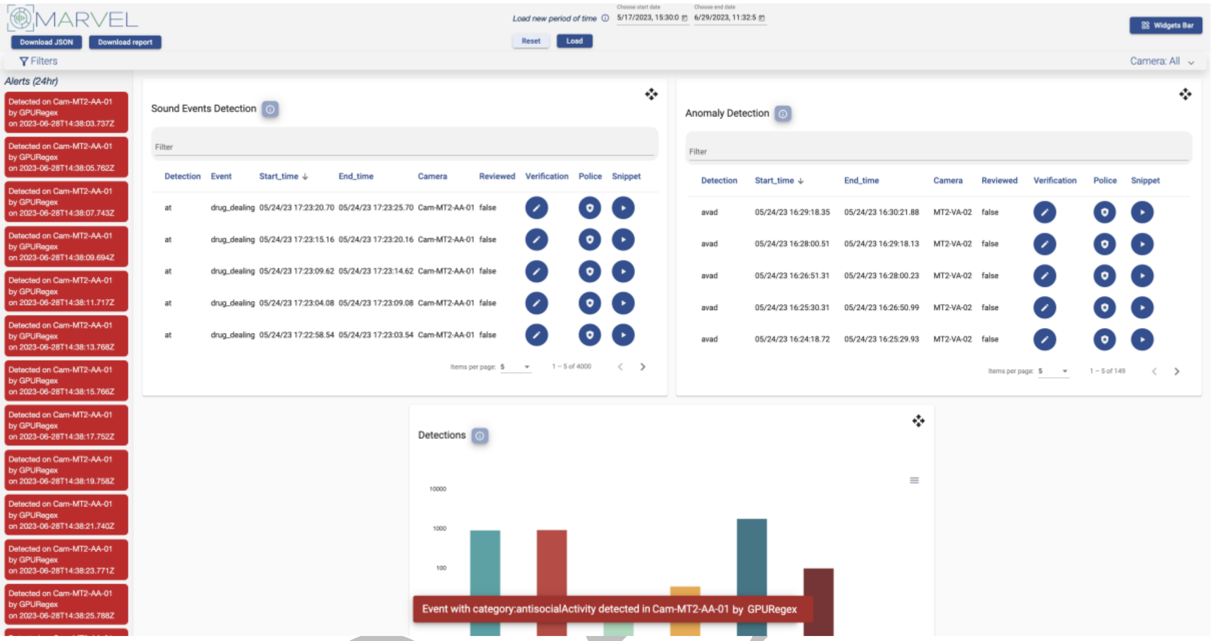

Sound Localisation Map Representation
+
−
Леополдов барутни магацин
Бастион св. Леополда
Молинаријева капија
Каменички пут
Leaflet | Map data © OpenStreetMap contributors
Frame and sound localisation player

Figure 71: Sound Localisation Map widget

5.5.3 Alerts

The Alerts widget (Figure 72) serves as a functionality catering for both real-time and historical data. The primary objective of this widget is to notify users whenever an alert is detected. When an alert occurs, a pop-up notification is displayed to the user, indicating the camera where the alert was detected, along with relevant information regarding the alert. In addition to real-time notifications, the Alerts widget also provides users with a summary of detected events from the past 24 hours. This feature allows users to review and gain insights into previously detected events.



Detected on Cam-MT2-AA-01 by GPURegex on 2023-06-28T14:38:03.737Z

Detected on Cam-MT2-AA-01 by GPURegex on 2023-06-28T14:38:05.762Z

Detected on Cam-MT2-AA-01 by GPURegex on 2023-06-28T14:38:07.743Z

Detected on Cam-MT2-AA-01 by GPURegex on 2023-06-28T14:38:09.694Z

Detected on Cam-MT2-AA-01 by GPURegex on 2023-06-28T14:38:11.717Z

Detected on Cam-MT2-AA-01 by GPURegex on 2023-06-28T14:38:13.768Z

Detected on Cam-MT2-AA-01 by GPURegex on 2023-06-28T14:38:15.766Z

Detected on Cam-MT2-AA-01 by GPURegex on 2023-06-28T14:38:17.752Z

Detected on Cam-MT2-AA-01 by GPURegex on 2023-06-28T14:38:19.758Z

Detected on Cam-MT2-AA-01 by GPURegex on 2023-06-28T14:38:21.740Z

Detected on Cam-MT2-AA-01 by GPURegex on 2023-06-28T14:38:23.771Z

Detected on Cam-MT2-AA-01 by GPURegex on 2023-06-28T14:38:25.788Z

Sound Events Detection

Detection	Event	Start_time ↓	End_time	Camera	Reviewed	Verification	Police	Snippet
at	drug_dealing	05/24/23 17:23:20.70	05/24/23 17:23:25.70	Cam-MT2-AA-01	false	✎	●	▶
at	drug_dealing	05/24/23 17:23:15.16	05/24/23 17:23:20.16	Cam-MT2-AA-01	false	✎	●	▶
at	drug_dealing	05/24/23 17:23:09.62	05/24/23 17:23:14.62	Cam-MT2-AA-01	false	✎	●	▶
at	drug_dealing	05/24/23 17:23:04.08	05/24/23 17:23:09.08	Cam-MT2-AA-01	false	✎	●	▶
at	drug_dealing	05/24/23 17:22:58.54	05/24/23 17:23:03.54	Cam-MT2-AA-01	false	✎	●	▶

Items per page: 5 1 – 5 of 4000

Anomaly Detection

Detection	Start_time ↓	End_time	Camera	Reviewed	Verification	Police	Snippet
avad	05/24/23 16:29:18.35	05/24/23 16:30:21.88	MT2-VA-02	false	✎	●	▶
avad	05/24/23 16:28:00.51	05/24/23 16:29:18.13	MT2-VA-02	false	✎	●	▶
avad	05/24/23 16:26:51.31	05/24/23 16:28:00.23	MT2-VA-02	false	✎	●	▶
avad	05/24/23 16:25:30.31	05/24/23 16:26:50.99	MT2-VA-02	false	✎	●	▶
avad	05/24/23 16:24:18.72	05/24/23 16:25:29.93	MT2-VA-02	false	✎	●	▶

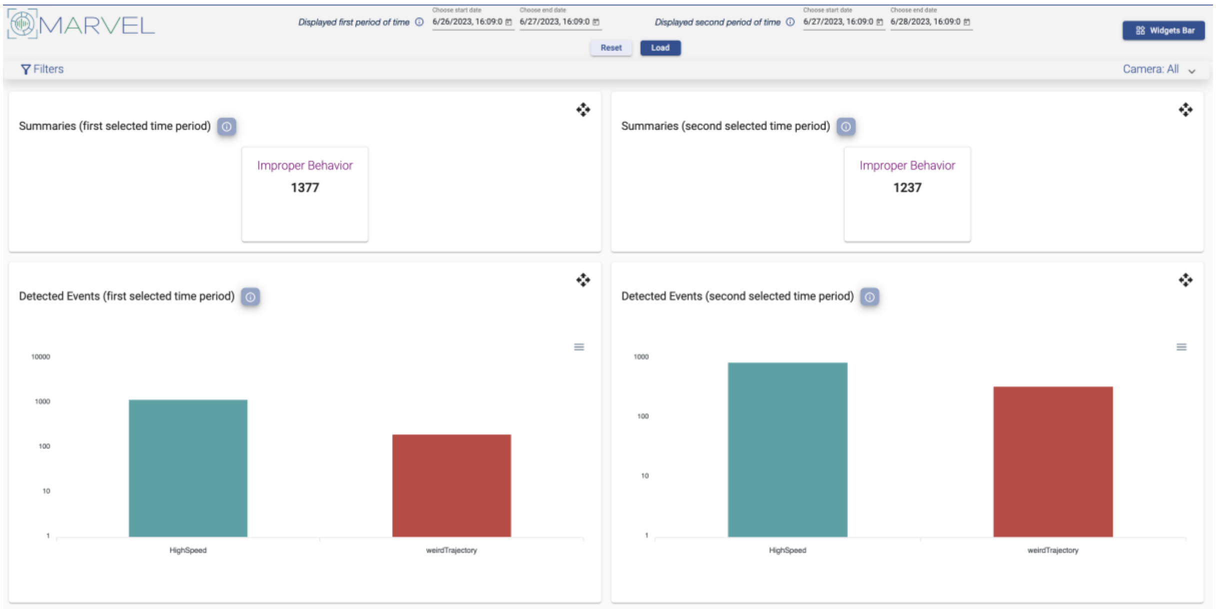
Items per page: 5 1 – 5 of 149

Event with category:antisocialActivity detected in Cam-MT2-AA-01 by GPURegex

Figure 72: Alerts widget

5.5.4 Comparison

The comparison functionality (Figure 73) allows users to view two separate time periods side by side, displaying the same widgets for each period. This feature enables users to compare and analyse the detected events within each time period. By examining the events across different time periods, users can identify hidden correlations between them. This comparison can be particularly useful for identifying patterns or trends, evaluating the effectiveness of educational campaigns, and drawing conclusions about their performance.



Displayed first period of time: Choose start date 6/26/2023, 16:09:0 – Choose end date 6/27/2023, 16:09:0

Displayed second period of time: Choose start date 6/27/2023, 16:09:0 – Choose end date 6/28/2023, 16:09:0

Camera: All

Summaries (first selected time period): Improper Behavior 1377

Summaries (second selected time period): Improper Behavior 1237

Figure 73: Comparison functionality

5.5.5 Download Data (JSON)

In the final version of the DMT, users have the option to download the visualised data in a JSON format. These downloadable data include any filters and preferences applied during the users' interaction with the DMT. By providing this functionality, users can easily retrieve and utilise the visualised data for further analysis or reference purposes.

5.5.6 Download PDF Report

In the final version of the DMT, users have the ability to download a PDF version of the dashboard view. This allows them to save the entire dashboard as a PDF file, providing a convenient way to store and use the dashboard according to their preference. Users can refer to the downloaded PDF version of the dashboard offline or share it with others as needed.

5.5.7 Word Cloud

The Word Cloud widget (Figure 74) in the DMT enables users to visualise the most common keywords and descriptions associated with the detected events. The data used for this widget are the output of the AAC component that create textual description with full sentences for audio segment. The caption of the component's outputs describes what is happening in the audio signal, for example, "people yelling while siren wails". In the word cloud, more frequently occurring captions are displayed with greater prominence, while less frequently used captions are shown with less emphasis. This visualisation provides a summary of the most prominent terms or topics within the data input.

By analysing the prominence of the captions in the word cloud, users can quickly identify the words that appear most frequently. This visualisation technique is particularly valuable for gaining insights into the main trends or keywords associated with the detected events and it allows users to extract meaningful information from the textual descriptions.

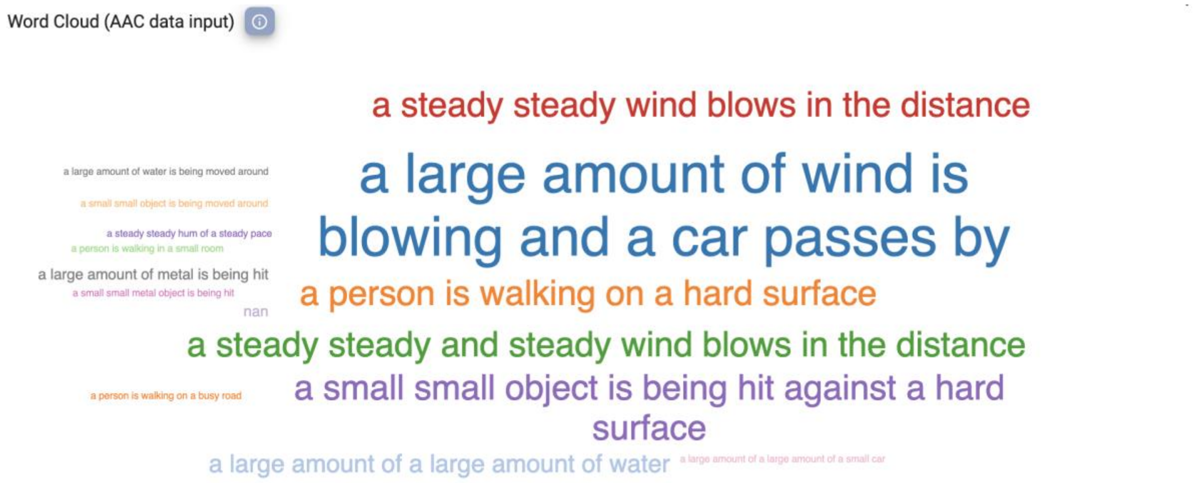

Word Cloud (AAC data input)
a steady steady wind blows in the distance
a large amount of water is being moved around
a small small object is being moved around
a steady steady hum of a steady pace
a person is walking in a small room
a large amount of metal is being hit
a small small metal object is being hit
nan
a large amount of wind is blowing and a car passes by
a person is walking on a hard surface
a steady steady and steady wind blows in the distance
a person is walking on a busy road
a small small object is being hit against a hard surface
a large amount of a large amount of water
a large amount of a large amount of a small car

Figure 74: Word Cloud widget

5.5.8 Audio Player

The audio player widget in the DMT (Figure 75) offers two main functionalities. Real-time audio streaming and playback of audio snippets related to specific events or detections. Users can listen to audio streams in real-time, monitoring ongoing audio data. They can also trigger playback of corresponding audio snippets when selecting specific events or areas for further analysis.

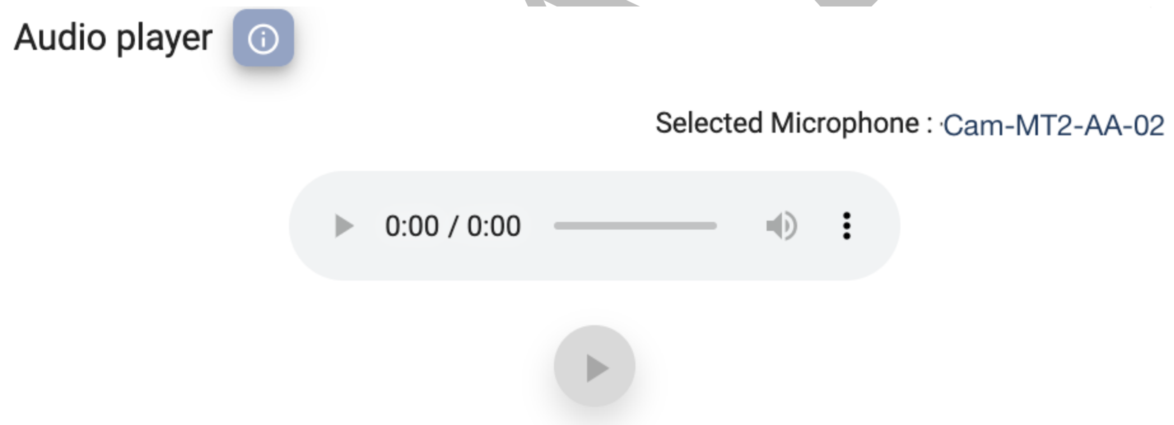
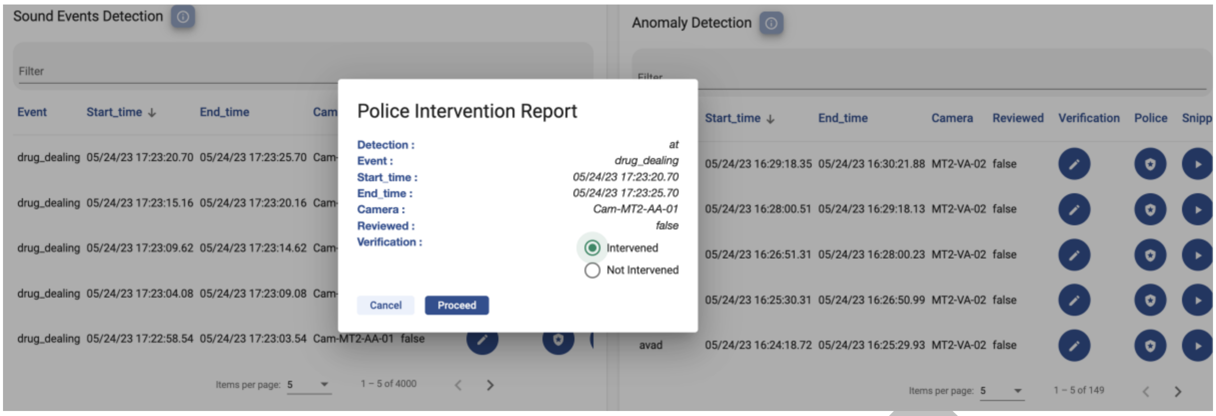


Figure 75: Audio player widget

5.5.9 Police Intervention

Police intervention widget (Figure 76) provides the ability for users, especially police authorities, to identify and flag specific events within DMT's visualisation interface. When observing a visualised event that requires police intervention, users have the option to mark or flag that event. By marking the event, users indicate its significance and the need for attention from authorities. This functionality allows users to annotate and highlight events they view as particularly important in their analysis and reporting.



Sound Events Detection

Filter

Event	Start_time ↓	End_time	Cam
drug_dealing	05/24/23 17:23:20.70	05/24/23 17:23:25.70	Cam-
drug_dealing	05/24/23 17:23:15.16	05/24/23 17:23:20.16	Cam-
drug_dealing	05/24/23 17:23:09.62	05/24/23 17:23:14.62	Cam-
drug_dealing	05/24/23 17:23:04.08	05/24/23 17:23:09.08	Cam-
drug_dealing	05/24/23 17:22:58.54	05/24/23 17:23:03.54	Cam-MT2-AA-01 false

Items per page: 5 1 – 5 of 4000

Police Intervention Report

Detection :	at
Event :	drug_dealing
Start_time :	05/24/23 17:23:20.70
End_time :	05/24/23 17:23:25.70
Camera :	Cam-MT2-AA-01
Reviewed :	false
Verification :	● Intervened ○ Not Intervened

Cancel Proceed

Anomaly Detection

Filter

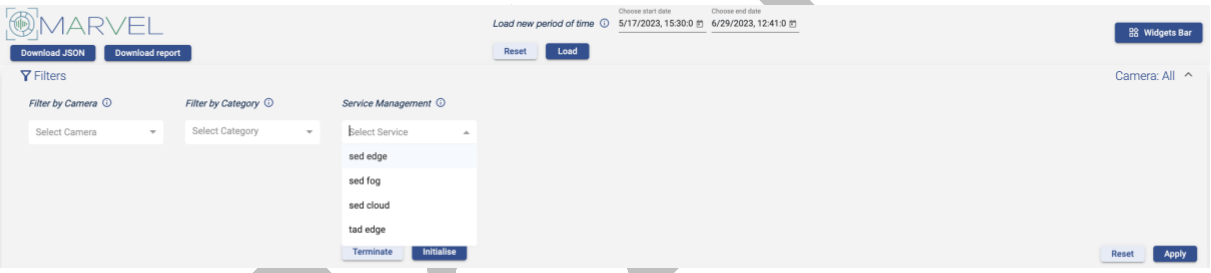
	Start_time ↓	End_time	Camera	Reviewed	Verification	Police	Snipp
	05/24/23 16:29:18.35	05/24/23 16:30:21.88	MT2-VA-02	false			
	05/24/23 16:28:00.51	05/24/23 16:29:18.13	MT2-VA-02	false			
	05/24/23 16:26:51.31	05/24/23 16:28:00.23	MT2-VA-02	false			
	05/24/23 16:25:30.31	05/24/23 16:26:50.99	MT2-VA-02	false			
avad	05/24/23 16:24:18.72	05/24/23 16:25:29.93	MT2-VA-02	false			

Items per page: 5 1 – 5 of 149

Figure 76: Police Intervention functionality

5.5.10 Service Management

In the final version of the DMT, users are empowered with service control capabilities (Figure 77). They can initiate or terminate specific services according to their needs. This functionality is made possible through the interaction between SmartViz and MARVDash via a REST API. This API allows SmartViz to retrieve information about the available services deployed within MARVDash. The users can then initiate and terminate selected services. This interaction provides the end users with the ability to control and manage services within the system.



Load new period of time Choose start date 5/17/2023, 15:30:0 Choose end date 6/29/2023, 12:41:0

Camera: All

Filter by Camera: Select Camera

Filter by Category: Select Category

Service Management: Select Service

- sed edge
- sed fog
- sed cloud
- tad edge

Terminate Initialise

Figure 77: Service Management functionality

Table 20, Table 21 and Table 22 map the DMTs' widgets to the use cases that were selected for R2. A full and detailed description of the available widgets, their features, graphic appearance, functionality, and operation is included in D4.6 *'Decision Making Toolkit-final version'*.

Table 20: DMT Widgets' mapping to R2 use cases of GRN

Widgets	GRN1	GRN2	GRN3	GRN4
Temporal Representation	-	YES	-	YES
Crowd Density Heatmap Representation	-	-	-	YES
Details Widget	YES	YES	YES	YES
Vehicle Trajectories	-	-	-	YES
Statistical Representation	-	YES	YES	YES
Video Player	YES	YES	YES	-
Real-time Map Representation	-	-	YES	-
Summaries	YES	YES	-	-
Weather information	YES	YES	YES	YES

Sound Localisation Map	-	-	-	-
Alerts	YES	-	-	-
Comparison	-	YES	-	-
Download Data (JSON)	YES	YES	YES	YES
Download PDF Report	YES	YES	YES	YES
Word Cloud	-	-	-	-
Audio Player	-	YES	YES	-
Police Intervention	-	-	-	-
Service Management	-	-	-	YES

Table 21: DMT Widgets' mapping to R2 use cases of MT

Widgets	MT1	MT2	MT3	MT4
Temporal Representation	YES	-	YES	YES
Crowd Density Heatmap Representation	YES	-	-	-
Details Widget	YES	YES	YES	YES
Vehicle Trajectories	-	-	-	YES
Statistical Representation	YES	YES	-	YES
Video Player	YES	YES	YES	YES
Real-time Map Representation	YES	-	-	-
Summaries	YES	YES	YES	-
Weather information	YES	YES	YES	YES
Sound Localisation Map	-	-	-	-
Alerts	YES	YES	YES	-
Comparison	YES	-	-	-
Download Data (JSON)	YES	YES	YES	YES
Download PDF Report	YES	YES	YES	YES
Word Cloud	-	YES	-	-
Audio player	-	YES	YES	YES
Police Intervention	YES	YES	YES	YES
Service Management	-	-	-	-

Table 22: DMT Widgets' mapping to R2 use cases of UNS

Widgets	UNS1	UNS2
Temporal Representation	-	-
Crowd Density Heatmap Representation	YES	-
Details Widget	YES	YES
Vehicle Trajectories	-	-
Statistical Representation	-	-

Video Stream Player	YES	-
Real-time Map Representation	-	-
Summaries	YES	-
Weather information	YES	YES
Sound Localisation Map	-	YES
Alerts	YES	-
Comparison	-	-
Download Data (JSON)	YES	YES
Download PDF Report	YES	YES
Word Cloud	-	-
Audio Player	YES	YES
Police Intervention	YES	YES
Service Management	-	-

5.6 Infrastructure

The available infrastructure that was employed for developing and integrating the MARVEL R2 integrated framework played an important role in the overall design, specification and final form of R2. This section presents the AV sources, the computational infrastructure and other key devices that were used in each pilot at the edge and fog tiers as well as the computational infrastructure used at the cloud tier that was provided by PSNC.

5.6.1 GRN Infrastructure

The GRN Infrastructure provided for the R2 integration consists of 3 IP cameras transmitting audio and video, 2 PCs simulating Edge layer computational nodes, 1 Nvidia Jetson Edge device and a server on loan from FORTH acting as the Fog computational node. GRN has also added a Jetson Apollo Dev Kit edge device and Arduino to control a prototype LED road sign. The following subsections describe each component in detail.

5.6.1.1 GRN Edge Tier

The GRN pilot includes three IP cameras; One camera is at Mgarr (a rural town in the north-west region) and the other two cameras are at Zejtun (an urban town close to the southern inner-harbour region. These cameras have no ability to process data at the Edge and can only transmit audio and video via an IP connection. The camera at Mgarr has slightly different specifications than the IP cameras at Zejtun. All three components were used in both GRN use cases (GRN3 and GRN4). Table 23 lists the specifications for each camera.

Table 23: Specifications for GRN IP Cameras

Specifications Type	Mgarr Camera	Zejtun Cameras
IP camera model	Safire 5MP Bullet Outdoor/Indoor IP Camera With POE	ANNKE outdoor 5MP PoE security cameras, Model I51DL, Lens: 2.8mm
Resolution	1920 x 1080 P	1920 x 1080 P
Frame Rate	25 fps	20 fps

Specifications Type	Mgarr Camera	Zejtun Cameras
Video Encoding	H.265	H.265
Audio Encoding	MP2L2	MP2L2
Audio Sampling Rate	32kHz	16kHz
Audio Stream Bitrate	64kbs	128kbps

GRN has a PC at the Mgarr location directly connected to the Mgarr IP Camera. **Table 24** shows the specifications of the GRN Edge PC 1 (GRN E1). This PC was used to carry out processing at the Edge.

Table 24: GRN Edge PC 1 (GRN E1) specifications

HW subsystem	Specifications
CPU	Intel Core i7-3770 @3.4GHz, 8 cores
GPU	GTX1650 4GB
Hard Drive	1 TB
RAM	32 GB

A second PC was added to the Zejtun location to directly connect to the Zejtun IP Cameras. Table 25 shows the specifications of the GRN Edge PC 2 (GRN E2). This PC was used to carry out processing at the edge and further avoid transferring of anonymised data.

Table 25: GRN Edge PC 2 (GRN E2) specifications

HW subsystem	Specifications
CPU	INTEL CORE I 7 12700
GPU	GEFORCE RTX 3080
Hard Drive	1 TB
RAM	CRUCIAL 16GB DDR4

The Jetson Apollo Dev Kit edge device is a development kit for the NVIDIA JetsonXavier NX. Table 26 shows the specifications of the Jetson Device. This device is connected to an Arduino nano which controls an LED board which alerts drivers.

Table 26: GRN Edge Jetson Device specifications

HW subsystem	Specifications
CPU	6-core NVIDIA Carmel ARM®v8.2 64-bit CPU 6MB L2 + 4MB L3 processor
GPU	NVIDIA Volta™ architecture with 384 NVIDIA CUDA®

	cores and 48 Tensor cores
Hard Drive	16 GB eMMC 5.1
RAM	8 GB 128-bit LPDDR4x @ 1600 MHz 51.2GB/s16 GB 128-bit LPDDR4x @ 59.7GB/s

All Edge devices are connected to 4G mobile routers.

During the MVP the original GRNEdge device was partially integrated and tested as part of the GRN edge infrastructure. The GRNEdge device suffered an unstable connection and frequent shutting down due to overheating, as reported in D5.2. From R1 onwards, the GRNEdge Device was replaced by the multiple devices described above. The latest setup resulted in an edge system that is stable most of the time, weatherproof and readily installed with industry standard streaming protocols and control.

Finally, for the specific needs of GRN1, a Traffic LED sign simulator was used comprising an Arduino Nano board that controls an LED board. The detailed specification of these devices are presented in Table 27 below.

Table 27: GRN Traffic LED sign simulator

HW components	HW subsystem	Specifications
Arduino Nano	**Memory**	2 KB of SRAM and 1 KB of EEPROM.
	Input and Output	14 digital pins on the Nano , 5V 40mA
	Communication	UART TTL (5V) serial communication, FTDI FT232RL serial communication over USB and the FTDI drivers
LED Lighting Development Tools NeoPixel Triple-Ring Board	**LEDS**	44 Thru-Hole LEDs - 66mm Diameter
	Input	DOUT Control data signal output, VDD Power supply LED - +3.5～+5.5, VSS Ground, DIN Control data signal input

5.6.1.2 GRN Fog Tier

GRN has set up a server on loan from FORTH as the GRN Fog node. This node processes anonymised streams for all the use cases. The Fog server is connected to internet via a wired connection.

Table 28 shows the specifications of the GRN Fog server node.

Table 28: GRN Fog server node specifications

HW subsystem	Specifications
CPU	AMD EPYC 7313P 16-Core Processor (32 Threads)
GPU	NVIDIA RTX A4000 16GB

Hard Drive	NVME drive 256GB running UBUNTU Server OS 2 x Enterprise SAMSUNG SSDs 3.5TB
RAM	252GB of RAM

5.6.2 MT/FBK Infrastructure

In the context of the R2 integration, the infrastructure for the MT pilot at the Edge has been extended and now includes the following devices:

- 8 IP cameras transmitting video.
- Raspberry Pi boards equipped with MEMS microphones installed at the sites of the MT2, MT3 and MT4 use cases.

Regarding the Fog layer, it has not been modified after R1 and 2 workstations provided by FBK continue to serve as the computational infrastructure resources at the Fog.

The infrastructure of the MT use cases is particularly complicated because:

- It involves the production networks of two different institutions.
- Access to the raw data collected by the surveillance cameras and microphones of MT is granted to FBK on the basis of a bilateral agreement between MT and FBK, but only with strict constraints.
- The devices in the surveillance network of MT cannot be part of the Kubernetes cluster.

Therefore, the Fog tier is split into two parts: the first part (MT F1) interfaces with the MT network via VPN, and the second one (MT F2) communicates with the MARVdash platform via the EdgeSec VPN component.

5.6.2.1 MT Edge Tier

The video sensing devices consist of two models of IP cameras:

- Basler BIP2
- Basler BIP.

Both models can record videos with a resolution of 1600x1200 pixels with the MPEG codec, while Basler BIP2 can reach a framerate of 12.5 fps and Basler BIP can reach framerate of 2 fps. Note that overall, 14 cameras are available for data recordings (11 Basler BIP2 and 3 Basler BIP). However, 2 cameras per use cases have been selected to demonstrate the MARVEL functionalities. The live video feeds will be streamed to the MTFOG1 via the VPN access provided by MT.

The audio sensing devices consist of Nano hub IFAG microphones mounted on a Raspberry Pi (RPi) 4B 8GB. In the R1 deployment, 2 audio devices were used in the use case MT-3 (Piazzale ex-Zuffo). The RPi acts as an Edge device and performs the initial processing of the captured audio signals: voice activity detection (VAD) and subsequent anonymisation (AudioAnony). Signals are recorded at 16kHz with 16-bits precision. The RPi implements an RTSP server that streams the anonymised audio data to the subscriber nodes (a rebound node in the FOG tier as explained later). The RPi also forwards MQTT messages from VAD.

Due to security and privacy issues, Edge nodes are not part of the MARVEL Kubernetes cluster.

Table 29 presents the AV sources and edge infrastructure devices used in the MT pilot for R2.

Table 29: MT Edge Infrastructure in R2

Sensing device *[e.g., camera, microphone, mobile phone]*	**Specifications**	**Location**	**Use Case**	**Stream Name**
Camera 1	Digital cameras - Basler BIP2-1600c-dn 12 fps, 1600 x 1200	Piazza Fiera	MT1	MT1-VA-01_PiazzaFiera3_VideoAnony
Camera 2	Digital cameras - Basler BIP2-1600c-dn 12 fps, 1600 x 1200	Piazza Duomo	MT1	MT1-VA-02_PiazzaDuomoNord_VideoAnony
Camera 3	Digital cameras - Basler BIP-1600c 2 fps, 1600 x 1200	Piazzale ex Zuffo	MT3	MT3-VA-01_PiazzaleExZuffo_Ingresso_via_SS._Cosma_e_Damiano_VideoAnony
Camera 4	Digital cameras - Basler BIP-1600c 2 fps, 1600 x 1200	Piazzale ex Zuffo	MT3	MT3-VA-02_PiazzaleExZuffo_Ingresso_via_Berlino_VideoAnony
Camera 5	Digital cameras - Basler BIP2-1600c-dn 12 fps, 1600 x 1200	Piazza S. Maria Maggiore	MT2	MT2-VA-01_Piazza_SMMaggiore_Obelisque_VideoAnony
Camera 6	Digital cameras - Basler BIP2-1600c-dn 12 fps, 1600 x 1200	Piazza S. Maria Maggiore	MT2	MT2-VA-02_Piazza_SMMaggiore_TrafficLight_VideoAnony
Camera 7	Digital cameras - Basler BIP2-1600c-dn 12 fps, 1600 x 1200	Piazza Dante	MT4	MT4-VA-01_Piazza_Dante_TrafficLight1_VideoAnony
Camera 8	Digital cameras - Basler BIP2-1600c-dn 12 fps, 1600 x 1200	Piazza Dante	MT4	MT4-VA-02_Piazza_Dante_TrafficLight2_VideoAnony
MEMS Microphone 1	Microphones IFAG-MEMS RPi 4B 4GB (for audio elaboration)	Piazzale ex Zuffo	MT3	MT3-AA-01_PiazzaleExZuffo_Ingresso_via_SS._Cosma_e_Damiano_AudioAnony
MEMS Microphone 1	Microphones IFAG-MEMS RPi 4B 4GB (for audio elaboration)	Piazzale ex Zuffo	MT3	MT3-AA-02_PiazzaleExZuffo_Ingresso_via_Berlino_AudioAnony
MEMS Microphone 3	Microphones IFAG-MEMS RPi 4B 4GB (for audio elaboration)	Piazza S. Maria Maggiore	MT2	MT2-AA-01_Piazza_SMMaggiore_Obelisque_AudioAnony
MEMS Microphone 3	Microphones IFAG-MEMS RPi 4B 4GB (for audio elaboration)	Piazza S. Maria Maggiore	MT2	MT2-AA-02_Piazza_SMMaggiore_TrafficLight_AudioAnony
MEMS Microphone 5	Microphones IFAG-MEMS RPi 4B 4GB (for audio elaboration)	Piazzale Dante	MT4	MT4-AA-01_Piazza_Dante_TrafficLight_AudioAnony

MEMS Microphone 5	Microphones IFAG-MEMS RPi 4B 4GB (for audio elaboration)	Piazzale Dante	MT4	MT4-AA-02_Piazza_Dante_Listone_AudioAnony

Furthermore, an Nvidia Jetson device was used at the edge (MT3 E2) for hosting an instance of VideoAnony for the needs of MT3. The device specifications can be found in Table 30.

Table 30: MT3 E2 Edge Jetson Device specifications

HW subsystem	Specifications
CPU	6-core NVIDIA Carmel ARM®v8.2 64-bit CPU 6MB L2 + 4MB L3 processor
GPU	NVIDIA Volta™ architecture with 384 NVIDIA CUDA® cores and 48 Tensor cores
Hard Drive	16 GB eMMC 5.1
RAM	8 GB 128-bit LPDDR4x @ 1600 MHz 51.2GB/s16 GB 128-bit LPDDR4x @ 59.7GB/s

5.6.2.2 MT Fog Tier (FBK)

The Fog tier is split into two separate nodes. As mentioned above, this configuration is required in order to comply with the requirements of MT in order to grant FBK access to the raw data. Therefore, one node (MTFOG1 in Figure 78) connects via VPN to the cameras and microphones and is not part of MARVdash. The other node (MTFOG2) is within the Kubernetes cluster and communicates with MTFOG1 via the local area network. In this way:

- access to MTFOG1 is restricted and controlled by standard FBK's security protocol;
- anonymised video and audio streams are made available to the nodes in the Kubernetes cluster.

Figure 78 illustrates the specific configuration that was implemented at the MT Fog layer.

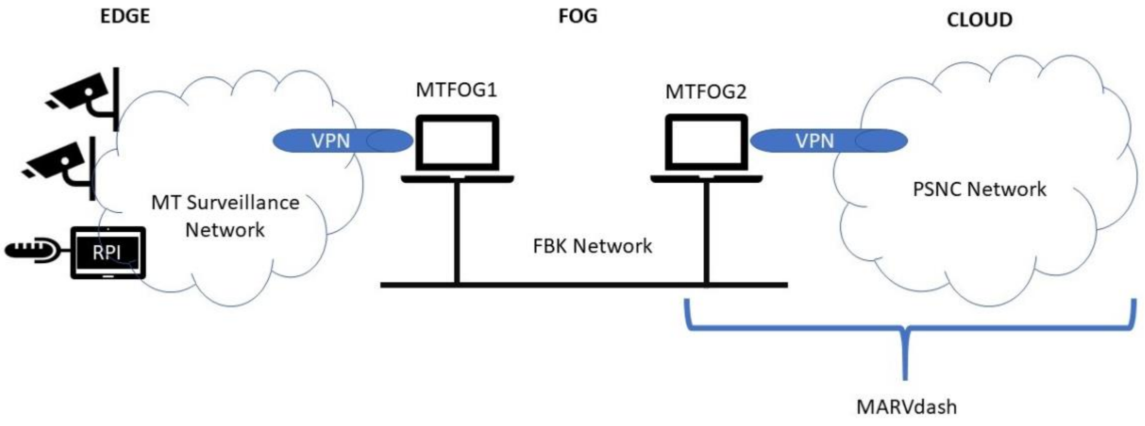


Figure 78: Infrastructure of the MT FOG, implemented at FBK premises

MTFOG1 (MT F1) is a workstation (WS) with a GPU. Since it is not part of the Kubernetes cluster this WS can be modified. The current workstation is equipped with a GPU GeForce RTX 2060 with 6GB of memory. It features a 12 core Intel(R) Core (TM) i7-8750H CPU @ 2.20GHz CPU with 32GB of RAM and a 500GB disk. MTFOG1 hosts:

- Videoanony.
- 2 RTSP clients to rebound the anonymised streams.
- 1 MQTT broker-client pair to rebound the VAD MQTT messages.

MTFOG2 (MT F1) is a workstation equipped with an Intel Xeon E5-1620 and Tesla K40 GPU with 11GB memory. The storage is 512 GB and the RAM is 20 GB. MTFOG2 is part of the Kubernetes cluster and hosts a DatAna fog node, StreamHandler, AV Registry, two instances of CATFlow for MT1, AAC and GPURegex for MT2 and two instances of CATFlow for MT4. It also hosts an RTSP server that receives the anonymised audio and video streams from MTFOG1, which resides outside of the MARVdash Kubernetes cluster.

The specification of MTFOG1 (MT F1) and MTFOG2 (MT F2) are presented in Table 31 below.

Table 31: MT Fog workstation specifications

HW subsystem	MTFOG1 (MT F1) Specifications (workstation PC)	MTFOG2 (MT F2) Specifications (workstation PC)
CPU	Intel Core i7-8750H @2.2GHz, 12 cores	Intel Xeon E5-1620
GPU	GeForce RTX 2060 6GB	Tesla K40 11GB
Hard Drive	500 GB	512 GB
RAM	32 GB	20 GB

5.6.3 UNS Infrastructure

In the context of R2, the UNS pilot hosts UNS1 (Drone experiment) and UNS2 (Localising audio events in crowds). The provided infrastructure consists of two tiers – Edge and Fog.

5.6.3.1 UNS Edge Tier

The Edge tier for UNS1 is based on the AVDrone component. It includes four Raspberry Pi (RPi) version 4 boards (UNS E2) and an Intel NUC mini PC (UNS E1). IFAG AudioHub Nano microphone boards are attached to the RPi v4 boards and are used for audio data processing on the ground. The setup of the drone includes an Intel NUC10i5FNH Mini PC, to which a GoPro camera is attached, which serves the needs of video anonymisation and data transmission to the Fog tier. It should be noted that the Mini PC supports Intel® Software Guard Extensions (Intel® SGX), which is a set of instructions that increase the security of application code and data, while it allows the deployment of EdgeSec TEE used for securing VideoAnony.

Furthermore, the following UAVs were used as part of the AVDrone:

- **DJI Matrice 600 Pro**[63] was used for demonstration purposes.

[63] https://www.dji.com/gr/matrice600-pro

- **DJI P4 Multispectral Drone** [64] was used for data collection due to its smaller size and less restrictive flight permissions.

Regarding the Edge tier for UNS2, an IFAG Audiohub – Nano 8 channel microphone board is used as a data capturing device. It is connected to the UNS E3 laptop at the Edge using Wi-Fi network. The microphone captures audio data using 8 spatially distributed sensors of the microphone array and transmits them to the laptop where the SELD and AudioAnony components are deployed. The user can set the sampling rate by selecting an option from a predefined set of values as well as the bitrate and number of channels for data capturing (up to 8 channels).

Default parameter values of the microphone board in UNS2 (Dual-PCB 8-microphones Audiohub – Nano board):

- Channels: 8
- Sampling rate: 48kHz
- Rate: 24-bit audio data

The specifications of the Edge devices at the UNS pilot for UNS1 and UNS2 are presented in Table 32.

Table 32: UNS Edge infrastructure specifications

HW subsystem	UNS E1 Specifications (Intel NUC10i5FNH Mini PC)	UNS E2 Specifications (Raspberry Pi version 4)	UNS E3 Specifications (Laptop HP ProBook 440 G9)
CPU	Intel Core i5-10210U @4.2GHz	Quad core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz	12th Gen Intel Core i7 - 1255U x 12.
GPU	Integrated	-	Integrated
Hard Drive	1TB M.2 NVMe	-	1TB
RAM	16 GB	2 GB	16 GB

5.6.3.2 UNS Fog Tier

AI models are executed at the Fog tier, including FedL which serves for model training in an ad-hoc mode. For this purpose, a SUPERMICRO SYS-7049A Server SuperWorkstation node is used, with the following specifications presented in Table 33.

Table 33: UNS Fog server specifications

HW subsystem	Specifications (SUPERMICRO SYS-7049A Server SuperWorkstation)
CPU	2 × Intel Xeon Silver 4110 2.1 GHz 8C/16T CPU
GPU	Nvidia TitanXP
Hard Drive	3 × 300GB SSD, 1 × 1TB SSD

[64] https://www.dji.com/gr/p4-multispectral

RAM	128 GB DDR4

Additionally, there is a local network storage appliance:

- QNAP TVS-682-8G with 4 × 3TB SATA3 disks under RAID protection, with NFS/SMB/S3 protocols enabled.

All infrastructure components are locally connected with multiple redundant 1G Ethernet links. UNS infrastructure is currently hosted “behind” an HTTP(S) proxy for Internet connectivity.

5.6.4 PSNC Cloud Infrastructure

The Cloud layer of the MARVEL framework is located on the PSNC premises. As mentioned in Section 4.5.1, Cloud resources come from two geographically distributed regions of PSNC - DCW and BST. Both areas are connected with a fast and dedicated (multiple 100Gbit links) network, allowing efficient data exchange between components running in different regions. Most of the cloud VMs (14 instances) are located in the DCW, which provides resources mainly for Kubernetes cluster nodes but also for general needs of the project, e.g., Zabbix monitoring, GitLab server. The BST is primarily used to store data in MARVEL Data Corpus using the CEPH storage backend. Summarising both regions, resources available exclusively for the MARVEL project include about 350 VCPUs, more than 500GB of RAM and about 1PB of storage.

Apart from that, after R1, PSNC dedicated one node of Eagle supercomputer, which joined the MARVEL Kubernetes cluster. It enabled the deployment of components that require constant access to GPU. This node has a direct connection with cloud infrastructure, allowing efficient two-sided communication between MARVEL containers running in the cloud layer (please see description in Section 4.6.1 and Figure 43: Connection between cloud and HPC infrastructure). Technically, it is a VM with Ubuntu 22.04 running on 16 CPUs and equipped with 64GB of RAM and direct access to 2x NVIDIA V100 GPUs (GPU passthrough technology).

6 Main challenges, issues encountered and resolution

During the R2 integration activities and especially in the context of Partial and End-To-End Integration Testing (Sections 3.8.2 and 3.8.3), issues and bugs regularly arose, as is expected during the integration phase of any complex system built collaboratively. These issues were documented and relevant action points were assigned to the corresponding partners for addressing them through updates to the respective components by using the MARVEL Issue Tracking System (Section 3.5) and shared online spreadsheets for reporting test results.

This section documents the most notable and frequent challenges and issues that were encountered as well as the actions that were taken to address them.

- **Manage inherent integration complexity.** As already mentioned in Section 3.1, the overall task of the MARVEL framework integration was found to be particularly challenging due to its inherent complexity resulting from its characteristics (i.e., number of integrated components, use cases, deployment layers, and pilot infrastructure nodes). The following measures were taken to address this:
    - Clear time plan with milestones from the outset (Section 3.1.1)
    - Organise regular Integration Meetings (Section 3.1.2)
    - Establish Integration Board (Section 3.1.3)
    - Implement an Issue Tracking System universally for all integration issues (Section 3.5)
    - Use of online shared templates and spreadsheets for organising and centrally documenting information related to integration (e.g., Figure 2, Figure 3, Figure 4, Figure 14)
    - Maintain clear, up-to-date and version-controlled documentation of system specifications, i.e. I/O interfaces and Data Models (Section 3.6)
    - Use of the MARVdash tool to facilitate the frequent re-deployment of more than 120 interconnected microservices across all infrastructure nodes and for all R2 use cases (Sections 3.7, 4.5.4).
- **Absence of dedicated component for AV data management.** From the early stages of R1, the lack of a foreseen technology in the MARVEL framework that could comprehensively manage AV data was recognised. The following actions were taken to address this:
    - Design of a universal framework relying on the RTSP protocol for live AV data transmission (Section 5.2.3).
    - Delegation of AV data handling to individual components that act as producers (e.g., VideoAnony, AudioAnony) or consumers (e.g., AI components) of live AV data streams. Each producer needs to implement its own RTSP server, and each consumer needs to implement its own RTSP client.
    - Used the ‘Camera’ Data Model to structure the AV source metadata (Section 5.3.1).
    - Introduced AV Registry (Section 4.1.3) as a new component in R1 for managing AV source metadata.
    - Added new features in StreamHandler (Section 4.3.3) to handle persistent AV data storage management and distribution. In R2, this functionality was further improved.
- **Complying with data privacy regulations and meeting security requirements.** Due to the nature of MARVEL comprising the collection of live AV data with potentially personal information and the need to access closed operational networks of public

authorities hosting AV sources, a series of requirements related to ethics, data privacy, and security were expected to be met. The Ethics Review Procedure that was carried out during the evaluation phase of the project identified 10 pre-grant ethics requirements and 10 post-grant ethics requirements in addition to the restrictions imposed by data owners (e.g., public authorities in Malta and Trento) This aspect of MARVEL was considered of increased significance and was challenging to achieve from a technical perspective. The following actions were taken to address this point:

  - Anonymisation of AV data (VideoAnony, AudioAnony) was positioned as close to the source as possible (i.e., at the edge or at the fog when this was not possible). Anonymising AV data close to the source reduces the risk of intercepting the raw data compared to the possibility of transferring to higher layers (e.g., cloud) for anonymisation. Also, this meant that only an anonymised form of the data would be distributed to other components for analysis, storage and viewing at higher layers.
  - Persistent storage of anonymised AV data close to the source. Even though it is foreseen to permanently store only AV data that have already been anonymised, further security measures were being considered to counteract possible risks of potential anonymisation failure. In this context, all StreamHandler (Section 4.3.3) instances were positioned at the fog layer, so that the storage of AV data occurs on devices that are at the premises and under the complete control of the data owners (pilots).
  - Secure communications between infrastructure nodes were established using EdgeSec VPN (Section 4.2.4)
  - Encrypted credentials were embedded in anonymisation services for accessing raw AV sources, i.e., in the GRN pilot the ‘secrets’ method of Kubernetes was used. Wherever the infrastructure allowed, increased security levels were achieved for protecting the credentials required to access AV sources by using the EdgeSec TEE component (Section 4.2.5); in R2, this was possible in UNS1.
  - Implemented customised network and fog infrastructure configuration at FBK for the MT pilot. For the needs of the MT pilot, FBK has been nominated as a Data Processor by the authorities of the Municipality of Trento. In order to gain access to the live raw AV data from the AV sources in the MT network, a customised solution involving dual servers on segregated networks at the fog layer and a series of VPN connections was applied. More details are provided in Sections 5.4.5, 5.4.6, 5.4.7, 5.4.8 and 5.6.2.2.
  - AI Training dataset preparation was carried out only by authorised personnel.
  - Continuous monitoring of the technical activities for ethics, legal, and privacy compliance by the MARVEL Ethics Board which consists of three independent members external to the MARVEL consortium, assisted by three consortium members.

- **Improving Stability.** During R1 integration activities, it became apparent that the system was facing stability issues and that a continuous, error-free operation could not be guaranteed. The main reasons for instability were (i) unstable streams from AV sources (e.g., short and long interruptions in transmission, corrupted data packets), (ii) limited computational resources on edge and fog devices, and (iii) network problems. In R2, considerable efforts were made to overcome these issues:
  - After recognising that signals from AV sources were often intermittent, it was requested from all owners of components that consume RTSP streams (CATFlow, SED, AT, AAC, SELD, VCC, AVCC, ViAD, AVAD, YOLO-SED,

StreamHandler) to implement error-handling mechanisms at the level of individual components. Each component implemented a mechanism that was better suited to the nature and the internal operation of the component in order to (i) avoid complete failure of the component when receiving interrupted or corrupted data streams, (ii) be able to recover and resume normal operation when the data stream is restored, and (iii) ensure that no repercussions are caused to other components when the input is not the one that is expected.

  - The tools used for monitoring the MARVEL framework were further extended in R2 by adding the **MARVdash monitoring tool** (based on Grafana, Prometheus and Loki) and the **MARVEL infrastructure monitoring tool** (based on Zabbix). These allowed the centralised live monitoring of multiple aspects of the running integrated framework and the Kubernetes cluster, including available computational resources, the resources consumed by running services, the log output of individual components, the status of MQTT brokers, all visualised via intuitive dashboard-based GUIs (Section 3.8.3).
  - In the GRN pilot, improved 4G routers and network bandwidth were secured for the edge devices. In addition, a second edge device with high computational resources was added to the Zejtun location that could handle the anonymisation of the two AV streams from that location and the partial AI computation for GRN2, GRN3, and GRN4. Furthermore, the fog server that had been used in R1 was replaced by another (provided as a loan by FORTH), which featured much higher computational resources (Section 5.6.1).
  - In the MT pilot, where there is a VPN tunnel connecting the network of the MT municipality and the network of FBK, actions were taken on both ends to configure a mechanism that automatically checks the VPN connection status and keeps it alive continuously. Furthermore, a more stable connection with increased bandwidth was secured for the VPN tunnel.
  - In the MT pilot, it was discovered that the 6 Raspberry Pi Edge devices hosting the MEMS microphones that were installed on-site at the locations of the MT2, MT3 and MT4 sites were affected by environmental temperatures and were malfunctioning when temperatures were very high (i.e.,. during the summer period). For this reason, FBK implemented a mechanism for remotely monitoring the Raspberry Pi boards and for being able to restart and reset the devices when a malfunction was observed.
  - In the MT pilot, thorough investigations took place by MT and FBK to discover and understand the root cause of continuing intermittence in the transmission of CCTV cameras and the reception of corrupted frames. It was concluded that the most likely explanation is the intermittent malfunction of the employed CCTV camera devices, which was not possible to rectify.

- **AV sources and infrastructure not available on a 24/7 basis.** It was not possible to have the streams from all AV sources available continuously due to limitations of the infrastructure used for processing/anonymising the streams (e.g., overheating) and due to conflicts with data collection procedures. Mitigation action:
  - This issue was addressed through a pro-active scheduling of the AV source availability, planning integration testing sessions at fixed time slots and using alternative staging/testing environments where possible.
- **Component deployment complications**. Component instances (services) were required to be deployed repeatedly (e.g., to provide updated service image for addressing issues) and on different infrastructure nodes. MARVdash facilitated this

process and FORTH had provided detailed instructions for the use of MARVdash. However, there were occasions where issues would arise, especially in cases of more complex services with specific requirements and/or constraints. Mitigation action:

  - FORTH continued to provide direct support to all component owners throughout the deployment process on a case-by-case basis to resolve all issues.
  - Relevant issues were created in the GitLab ITS when necessary.
  - Issues were also discussed and resolved during integration meetings and E2E testing sessions.

- **Integration testing complexity.** We have found that the testing process was encumbered by multiple component and infrastructure co-dependencies at run-time during the execution of experiments. Mitigation actions:
  - 30 intensive E2E testing sessions were organised during R2 activities to ensure sufficient testing of all use cases (Section 3.8.3).
  - Parallelisation of tests in each session to make efficient use of time.
  - Promotion of direct chat communication using a common channel for integration issues.
  - Pro-activeness: The necessary preparations that were needed in advance of each E2E test were communicated through email, testing documentation spreadsheets and issues in the ITS (Section 3.5).
- **Service exposure.** In some cases, particular attention and specialised handling were required to expose services that could not reached by others. Notable examples:
  - The RTSP proxy services in the MT use cases were deployed outside of the MARVEL Kubernetes cluster and a secure VPN connection had to be configured to make the streams of anonymised AV data available to other components of the Kubernetes cluster.
  - The DFB Kafka service was exposed using the nodeport method so that it could be reached from the Data Corpus, which was deployed on a different VM at the PSNC HPC.
  - The NGINX [65] Reverse Proxy service has been employed and configured so that some services could be reachable from services outside of the Kubernetes cluster, which proved to very useful for debugging purposes during development and testing.
- **Fine-Tuning AV data streams.** Based on preliminary test results, the AV data stream output of VideoAnony and AudioAnony was fine-tuned to serve the needs of AI components for improved performance.
- **Inconsistencies in inference result data**. On several occasions, inconsistencies were observed in the raw AI inference results produced by AI components. A frequent issue was the incorrect formatting of the date/time field values or missing fields. Nevertheless, the persistent and up-to-date documentation of all foreseen data models in the project's repository (Sections 3.6, 5.3.3), the revision of the documentation following test execution and the opening of relevant issues in the ITS (Section 3.5) when problems were observed assisted in significantly reducing such occasions.
- **Issues with REST API calls**. On some occasions, mismatches in service names, open ports or syntax in the API calls caused some disruption, but all these issues were quickly addressed and resolved thanks to the thorough documentation that was maintained for all I/O interface specifications and service deployment configuration (Sections 3.6, 3.7, 5.2 and Figure 14).

[65] https://www.nginx.com/

- **Shifting AI computation closer to the edge layer.** In some cases, it has been difficult to shift AI computation closer to the edge due to limited computational resources in available edge and fog devices. Mitigation actions:
    - GPU support was added in AI components (CATFlow, VideoAnony, ViAD, AVAD, VCC, SED, AT, AAC, SELD, YOLO-SED, GPURegex) to make optimal use of available resources and increase performance due to GPU-enabled multi-threading.
    - Additional and more powerful edge devices were secured (GRN E2, GRN E3, UNS E3), while a new more powerful GRN fog server was installed (on loan from FORTH). (Section 5.6)
    - Adaptations to the system architecture design for each use case to make optimal use of available resources.
- **Deploying docker containers of components with GPU usage on the Jetson device.** In the case of GRN1, this proved to be a highly demanding task due to a conflict between specific firmware versions of the Nvidia Jetson and Kubernetes tools related to GPU usage. Eventually, this issue was resolved, following experiments with various configurations coupled with tests, offline communications between the involved partners and tracking the issue through the MARVEL ITS (Section 3.5).

7 R2 main achievements and contribution to MARVEL goals

7.1 R2 main achievements

The main accomplishments of R2 are summarised below:

- **R2 addresses the complete set of the MARVEL use cases (10).**
- **R2 integrates the complete set of the MARVEL components (32).**
- **Reinforcement of the coherent and versatile architecture**. The design of the R2 ‘AI Inference Pipeline’ architecture combines a coherent core, which remains consistent across all use cases and interchangeable, configurable endpoints that can be deployed at the Edge and Fog in response to specific use case requirements. This was further reinforced and demonstrated in additional use cases in R2 and supported additional components that were introduced and integrated in R2.
- **Parallel operation of multiple component instances**. In each R2 use case, the data streamed from each AV source was processed by a distinct instance of each AI component (service), which increased integration complexity but achieved a closer approximation real-life operating conditions. In R2, this was further scaled up due to the presence of additional use cases, components and infrastructure nodes.
- **Improved computational infrastructure.** In R2, the available infrastructure was upgraded by introducing two new edge devices in GRN (PC and Jetson), by replacing the GRN fog server by another of higher specifications, by adding five (5) new edge devices in MT (1 Jetson and 5 Raspberry Pi devices) and by adding one more edge device in UNS (PC). All the new edge and fog devices apart from the RPis support GPU computation. Furthermore, 6 new VMs were added to the cloud infrastructure. In R2, 6 of the total 18 VMs allocated for MARVEL were attached to the Kubernetes cluster used for the deployment of services that support the inference pipeline in the R2 use cases. One of these VMs belongs to the PSNC Eagle cluster and supports high-performance GPU-based computation.
- **Improved overall system stability**. In R2, particular focus was paid to improving the overall system stability. This was achieved by implementing error-handling and recovery mechanisms in components that consume AV data streams, developing and employing dedicated centralised GUI-based monitoring tools (MARVdash monitoring tool, MARVEL infrastructure monitoring tool) for improving the infrastructure and its configuration, enhancing network performance and implementing mechanisms that allow manual recovery in cases of instability (e.g., remote monitoring and reset of AudioAnony in MT pilot when overheating).
- **Operation with additional parallel real-world live (real-time) data streams**. Multimodal sensors (CCTV IP cameras, AV Drone with GoPro camera, MEMS microphones) were deployed at the Edge and simultaneously streamed the captured AV data to anonymisation services (VideoAnony, AudioAnony, VAD) that processed it in real time and served the output to multiple other component endpoints. In total, 17 AV sources were used in R2 compared to 9 AV sources in R1.
- **More extensive use of the unified framework for AV data distribution**. The RTSP protocol was adopted universally for all AV data streaming needs in the MARVEL project. Each component that needs to access an AV data source can directly connect to it and consume the stream. This process is further facilitated by the AV Registry component that disseminates AV source metadata. In R2, the AV data distributions framework was scaled up and was used more extensively in more use cases, by more components and involved additional AV sources.

- **Reliable and efficient AI inference result data collection from all EFC layers.** R2 reinforces the web of multiple data collection endpoints. Distinct DatAna MQTT message brokers are deployed at each infrastructure node where AI components are operating to directly receive AI inference results. The inference results are propagated to higher EFC layers by DatAna NiFi to be persistently stored at the DFB and consumed by SmartViz. In R2, DatAna was deployed on all new infrastructure nodes that were added for the needs of the R2 use cases and supported the data models of the new AI components that were introduced in R2 (AAC, SELD, YOLO-SED, RBAD, GPURegex).
- **Increased data security**. The use of the EdgeSec VPN component was further extended covering all new infrastructure nodes that were added, ensuring that all communications within the MARVEL Kubernetes cluster are secure. In addition, the EdgeSec TEE component was added in UNS1 for R2 to provide elevated protection to raw AV data and grant it exclusively to authorised components (VideoAnony) running within a trusted execution environment.
- **Improved persistent storage of AV data for reviewing detected events.** Anonymised live AV data streams are persistently stored by the StreamHandler component on infrastructure that is managed by the data owner or an authorised party and made available to MARVEL users in association with detected events. In R2, the operation of StreamHandler was improved in terms of stability and performance, while it was connected to the DFB in order to be able to generate AV segments that directly correspond to the inference results that arrive at the DFB.
- **Extensive and improved implementation of advanced deployment support and management.** MARVdash was used to facilitate the deployment of all components at the MARVEL Kubernetes clusters that unified infrastructure nodes across all EFC layers and pilot locations. In R2, the new MARVdash monitoring tool was integrated, the service creation was further optimised, and an API was established that can be used for controlling the status of multiple services (e.g., start, stop, etc).
- **Reinforcement of adaptive deployment for meeting end-user requirements (MT use cases)**. Through the specific implementation and deployment configuration that was performed for the MT use cases, MARVEL demonstrated that it can be significantly customised and propose creative solutions in response to demanding constraints set by the end-users. In the MT use cases, the hard requirements associated with data privacy and security were satisfied through a tailored solution comprising specialised infrastructure configuration and the deployment of necessary auxiliary services for overcoming data access barriers, while preserving high data privacy and security standards. The solution that was devised can be leveraged to address the needs of other MARVEL use cases as well as potential customer needs in future business cases. In R2, this was demonstrated in two additional use cases (MT2, MT4), while additional AV sources and edge devices were added and supported.
- **Further AI training with datasets originating from specific use case data sources**. R2 contributed to the further collection of raw and annotated datasets from the pilot sites to be used in AI training of the MARVEL components and ensure that the resulting AI models achieve increased performance and accuracy in the context of the addressed use cases.
- **Performance optimisation.** More components with GPU support were integrated (ViAD, AVAD, VCC, SED, AT, AAC, SELD, YOLO-SED).

- **Improved UI/UX**. The UI/UX elements of SmartViz were further elaborated and refined, while new elements were introduced to provide an enhanced Decision-Making Toolkit that fits the needs of the additional use cases addressed by R2.
- **More extensive use of new data management features**. DatAna provides inference results transformation to selected SDM-compliant homogeneous data models, while DFB provides services for fusing selected inference result types and updating the verification status of all results according to the user input through SmartViz for AI training purposes (labelling). This process was applied in 5 additional use cases and for additional AI components and was further fine-tuned and streamlined.
- **Implementation of mechanisms for benchmarking MARVEL**. A benchmarking framework and elements in the system that can be used for performance measurements (e.g., data model timestamps specification and implementation for tracking the latency in inference result routes) have been implemented.
- **More extensive use of management and CI/CD procedures**. The organisational mechanisms and CI/CD tools and methods that had been introduced in R1 (e.g., weekly integration meetings, Integration Board, shared online templates, Issue Tracking System, Version Control System, documentation repository, code repository, quality assurance) matured further and were applied more extensively in R2.

Table 34 demonstrates a summary of the incremental scale-up that was achieved between consecutive releases of the MARVEL framework.

Table 34: Comparison of main achievements in each MARVEL framework release

Achievements	MVP (M12)	R1 (M18)	R2 (M30)
Integrated components	12	26	32
Pilots	1	3	3
Use Cases	1	5	10
Infrastructure Nodes at E/F/C layers	E: 1 F: 1 C: 2 VMs (1 for k8s)	E: 4 (3 for k8s) F: 4 (3 for k8s) C: 12 VMs (2 for k8s)	E: 12 (6 for k8s) F: 4 (3 for k8s) C: 18 VMs (6 for k8s)
Input AV data	Pre-recorded	Live streams	Live streams
AV data distribution framework	Ad-hoc	RTSP protocol, AV Registry, StreamHandler	RTSP protocol, AV Registry, StreamHandler
Handling intermittent AV streams	N/A	None	Error-Handling and recovery implemented at consumer component level
AV sources per pilot	N/A	GRN: 3 MT: 4 UNS: 2	GRN: 3 MT: 11 UNS: 3
Component instantiations	Single	Multiple	Multiple
Deployment with MARVdash	Partial	Complete	Complete
Edge Computation	No	Partial	Complete
GPU availability in infrastructure	No	Partial	Complete

Component support for GPU acceleration	CATFlow, VideoAnony	CATFlow, VideoAnony	CATFlow, VideoAnony, ViAD, AVAD, VCC, SED, AT, AAC, SELD, YOLO-SED, GPURegex
Issue Tracking System	No	Yes	Yes
I/O Interface and Data Model specification documentation (with version control)	No	Yes	Yes
Centralised monitoring tools	No	Partial	Complete
Stability improvements	No	Partial	Complete
Inference result data model	Custom	Compliant to Smart Data Models standards	Compliant to Smart Data Models standards

7.2 Contribution of R2 to MARVEL goals

The overarching goal of MARVEL is to deliver an Edge-to-Fog-to-Cloud (E2F2C) framework that operates in real-world environments, processing large volumes of captured AV data, enabled by multi-modal perception and intelligence.

More specifically, with regard to the project's objectives, analysed in the DoA, the delivery of the final version of the MARVEL Integrated framework (R2) contributes to the achievement of each objective by supporting the respective Enablers in the way that is described below.

> *Objective 1: Leverage innovative technologies for data acquisition, management and distribution to develop a privacy-aware engineering solution for revealing valuable and hidden societal knowledge in a smart city environment*

MARVEL R2 contributes to the achievement of this objective by bringing forward a concrete definition of a comprehensive 'AI Inference Pipeline' that incorporates integrated solutions for **E1 - Data Capturing** (MEMS microphones, AV Drone, CCTV cameras, Go Pro camera), **E2 – Data fusion and management** (DatAna, DFB, StreamHandler, HDD, Data Corpus) and **E3 – Privacy preservation and assurance** (AudioAnony, VideoAnony, EdgeSec VPN, EdgeSec TEE, DatAna NiFi TLS security, DFB Keycloak, AV data persistent storage near the source – Fog layer, AI inference result decoupling from AV data).

> *Objective 2: Deliver AI-based multimodal perception and intelligence for audio-visual scene recognition, event detection and situational awareness in a smart city environment*

MARVEL R2 contributes to the achievement of this objective by integrating and testing a series of AI components (CATFlow, TAD, VAD, SED, AT, VCC, AVCC, ViAD, AVAD, AAC, SELD, YOLO-SED, RBAD), including AI components with inherent multimodal audio-visual processing capabilities (AVAD, AVCC, YOLO-SED) and **E1 - Multimodal federated learning approaches** (FedL), producing inference results for enhanced understanding of smart city environments (**E2 - Multimodal representations**). Additionally, the output of individual AI components with unimodal processing capabilities is intelligently combined at the MARVEL Decision Making Toolkit (SmartViz) to provide a multimodal situational awareness and reveal hidden events and relationships (**E3 - Multimodal intelligence**).

> *Objective 3: Break technological silos, converge very diverse and novel engineering paradigms and establish a distributed and secure Edge-to-Fog-to-Cloud (E2F2C) ubiquitous computing framework in the big data value chain.*

MARVEL R2 contributes to the achievement of this objective by delivering an operational solution that is fully distributed and deployed on an E2F2C infrastructure, enabled by (i) **E1 – HPC infrastructure and resource management** (PSNC HPC infrastructure, management and orchestration), (ii) E2 **- Distributed and optimised DL models deployment** (DynHP, FedL), (iii) **E3 - Secure and distributed computing framework** (MARVdash, Kubernetes cluster, EdgeSec VPN) and (iv) **E4 - Complex decision-making and insights** (SmartViz).

> *Objective 4: Realise societal opportunities in a smart city environment by validating tools and techniques in real-world settings.*

MARVEL R2 contributes to the achievement of this objective by (i) elaborating and addressing ten (10) use cases (GRN1, GRN2, GRN3, GRN4, MT1, MT2, MT3, UNS1, UNS2) in the context of three (3) pilots (GRN, MT, UNS) (**E1-Real-life societal experiments definition**), (ii) specifying a methodology and toolset for Quality Assurance and technical validation (Section 3.8) of the framework's operation (**E2-Experimentation variables**) and (iii) deploying and testing the R2 framework prototype on the real-life infrastructure hosted at the three pilot sites for five use cases (**E3 - Experimentation execution**).

> *Objective 5: Foster the European Data Economy vision and create new scientific and business opportunities by offering the MARVEL Data Corpus as a free service and contributing to BDVA standards*

MARVEL R2 contributes to the achievement of this objective by (i) integrating the components and infrastructure that allow the collection (VideoAnony, AudioAnony, AV Drone, RTSP protocol, AV Registry) and persistent storage (StreamHandler, Data Corpus) of AV data (**E1-Create an audio-visual dataset**), (ii) specifying, implementing and testing the necessary interfaces for providing access to these AV data (**E2-Sharing of the dataset**)

Finally, R2 provides a functional platform that can be used for the evaluation of technical KPIs that are linked to the above Objectives, concerning advanced data management, data privacy, novel algorithms, algorithmic accuracy and performance, facilitation of complex decision-making and validation strategy (the associated KPIs were also elaborated in D6.1 and D6.3). R2 provides the mechanisms for a thorough technical evaluation of the MARVEL framework through benchmarking activities in T5.4, while end-user feedback and evaluation will occur during upcoming T6.3 activities. The results of these evaluations will be detailed in the respective deliverables D5.5 and D6.4.

8 Future work

As part of the R2 integration activities, we have identified the following areas for future improvement:

- Implementation of a central mechanism for configuring the entire framework deployment (e.g., specify components, target deployment locations, input/output of each component instance, associations between components).
- Connection to third-party data sources (e.g. AI components that do not belong in the MARVEL framework, external databases, etc.) and/or third-party data visualisation tools.
- Implement long-term data analytics approaches for post-processing aggregated inference results that refer to long time periods after building relevant datasets.
- Improvement of CI/CD procedures (e.g. automated deployment pipelines connected to source code repositories).
- Implementation of E2E test automations.
- Live re-configuration of deployed running services (e.g., start, pause, change input/output) possibly via implementation of control signals through message brokers.
- Further improvement of system stability, particularly in relation to AV data stream access.
- Broader use of DynHP and FedL techniques in combination with AI components.
- Further automation of the pipeline involving AI training tasks and the transfer of trained AI models for inference operation.
- Further improvement of AV data distribution management, e.g., by implementing proxy server that can redirect each component to the data stream of the corresponding AV source.

9 Conclusions

This report presented the **final version of the MARVEL Integrated framework (Release 2 or R2)**, whose development and integration were based on the further elaboration of previous work on the framework's architecture and use case definition and the experience gained from the delivery of the previous MARVEL framework releases, i.e., MVP (D5.1) and R1 (D5.3).

The integration plan, methodology, and overall design approach adopted for the development and delivery of R2 were presented in detail. The scope and objectives of R2 were elaborated, followed by a description of the use cases that were addressed by R2, coupled with technical decisions regarding infrastructure, involved components, and integration of technologies into a unified, end-to-end platform. All components that have been integrated into R2 were presented and a detailed account was given on the design of the architectural configuration of the MARVEL framework. The specific architecture that was implemented for each of the selected R2 use cases was explained and all involved UI/UX elements, I/O interfaces and data models were documented. The specific infrastructure that was employed and the R2 deployment procedure were presented. Finally, the main achievements and contribution of R2 to the overall MARVEL goals were discussed, followed by an outline of the possible next steps for a potential future release of the MARVEL integrated framework.

This document can be used as a basis and reference for planned activities, deliverables and milestones, most notably: a) the planned benchmarking sessions within Task 5.4, to be part of deliverable D5.5 *'Technical evaluation and progress against benchmarks'*, b) the public events, showcases and info days where the R2 will be demonstrated to the public, c) the pilot execution and evaluation, deliverables D6.3 *'Demonstrators execution – final version'*, D6.4 *'Final assessment report and impact analysis'*, and d) the upcoming deliverable D5.7 *'MARVEL's framework large scale deployment'*. Especially for the aforementioned deliverable D5.7, the present deliverable D5.6 can serve as the main input, providing the blueprints of the MARVEL framework design and deployment possibilities for consideration in larger-scale deployment scenarios.

Appendix A: R2 Technical Validation Test Report

Test ID	Component A	Direction	Component B	I/O ID	Description	Test Date	Result
GRN1: Safer Roads							
T-GRN1.01	AV Registry	→	StreamHandler	2	StreamHandler GRN F2 received Camera objects from AV Registry GRN F2 for ids Cam-GRN-VA-01, Cam-GRN-VA-01-Audio	15-06-23	SUCCESS
T-GRN1.02	AV Registry	→	SmartViz	2	SmartViz C1 received Camera objects from AV Registry GRN F2 for ids Cam-GRN-VA-01, Cam-GRN-VA-01-Audio	15-06-23	SUCCESS
T-GRN1.03	CCTV Camera	→	YOLO-SED	3	YOLO-SED GRN1 E3 received RTSP stream from camera with id Cam-GRN-CCTV-01	15-06-23	SUCCESS
T-GRN1.04	CCTV Camera	→	VideoAnony	3	VideoAnony GRN E1 received RTSP stream from camera with id Cam-GRN-CCTV-01	15-06-23	SUCCESS
T-GRN1.05	VideoAnony	→	StreamHandler	3	StreamHandler GRN F2 received RTSP stream from VideoAnony GRN E1 for ids Cam-GRN-VA-01, Cam-GRN-VA-01-Audio	15-06-23	SUCCESS
T-GRN1.06	YOLO-SED	→	DatAna	4	YOLO-SED GRN1 E3 published raw inference results to DatAna MQTT GRN E3	15-06-23	SUCCESS
T-GRN1.07	Arduino Proxy	←	DatAna	5	ArduinoProxy GRN1 E3 consumed raw inference results from YOLO-SED GRN1 E3 for Cam-GRN-VA-01, Cam-GRN-VA-01-Audio by subscribing to the "YOLO-SED" topic of DatAna MQTT GRN E3.	15-06-23	SUCCESS
T-GRN1.08	Arduino Proxy	→	LED Sign Control	N/A	ArduinoProxy GRN1 E3 sent control commands over serial protocol to the LED Sign Control script on Arduino	15-06-23	SUCCESS
T-GRN1.09	StreamHandler	↔	SmartViz	9	SmartViz C1 made REST API requests to StreamHandler GRN F2 for AV content related to inference results generated by YOLO-SED. StreamHandler returned the urls for requested AV content.	30-06-23	SUCCESS
T-GRN1.10	StreamHandler	→	SmartViz	10	SmartViz C1 received the AV files for requested inference results (YOLO-SED) from StreamHandler GRN F2	30-06-23	SUCCESS
T-GRN1.11	DatAna	→	DFB	7	DFB C1 received inference results (YOLO-SED) from DatAna C1	15-06-23	SUCCESS
T-GRN1.12	DFB	↔	SmartViz	8	SmartViz C1 received inference results (YOLO-SED) from DFB Kafka / Elastic Search C1 DFB C1 received inference verification messages (YOLO-SED) from SmartViz C1	21-06-23	SUCCESS
GRN2: Road user behaviour							

Test ID	Component A	Direction	Component B	I/O ID	Description	Test Date	Result
T-GRN2.01	AV Registry	→	CATFlow	**2**	CATFlow GRN E1, E2, F2 received Camera objects from AV Registry GRN F2 for ids Cam-GRN-VA-01, Cam-GRN-VA-02, Cam-GRN-VA-03	24-25/05/2023	SUCCESS
T-GRN2.02	AV Registry	→	SED	**2**	SED GRN2 E1, E2, F2 received Camera objects from AV Registry GRN F2 for ids Cam-GRN-VA-01-Audio, Cam-GRN-VA-02-Audio, Cam-GRN-VA-03-Audio	24-25/05/2023	SUCCESS
T-GRN2.03	AV Registry	→	StreamHandler	**2**	StreamHandler GRN F2 received Camera objects from AV Registry GRN F2 for ids Cam-GRN-VA-01, Cam-GRN-VA-01-Audio, Cam-GRN-VA-02, Cam-GRN-VA-02-Audio, Cam-GRN-VA-03, Cam-GRN-VA-03-Audio	24-25/05/2023	SUCCESS
T-GRN2.04	AV Registry	→	SmartViz	**2**	SmartViz C1 received Camera objects from AV Registry GRN F2 for ids Cam-GRN-VA-01, Cam-GRN-VA-01-Audio, Cam-GRN-VA-02, Cam-GRN-VA-02-Audio, Cam-GRN-VA-03, Cam-GRN-VA-03-Audio	24-25/05/2023	SUCCESS
T-GRN2.05	VideoAnony	→	CATFlow	**3**	CATFlow GRN E1, E2, F2 received RTSP stream from VideoAnony GRN E1, E2, F2 for ids Cam-GRN-VA-01, Cam-GRN-VA-02, Cam-GRN-VA-03	24-25/05/2023	SUCCESS
T-GRN2.06	VideoAnony	→	SED	**3**	SED GRN2 E1, E2, F2 received RTSP stream from VideoAnony GRN E1, E2, F2 for ids Cam-GRN-VA-01-Audio, Cam-GRN-VA-02-Audio, Cam-GRN-VA-03-Audio	24-25/05/2023	SUCCESS
T-GRN2.07	VideoAnony	→	StreamHandler	**3**	StreamHandler GRN F2 received RTSP stream from VideoAnony GRN E1, E2, F2 for ids Cam-GRN-VA-01, Cam-GRN-VA-01-Audio, Cam-GRN-VA-02, Cam-GRN-VA-02-Audio, Cam-GRN-VA-03, Cam-GRN-VA-03-Audio	24-25/05/2023	SUCCESS
T-GRN2.08	CATFlow	→	DatAna	**4**	CATFlow GRN E1 published raw inference results to DatAna MQTT GRN E1. CATFlow GRN E2 published raw inference results to DatAna MQTT GRN E2. CATFlow GRN F2 published raw inference results to DatAna MQTT GRN F2	24-25/05/2023	SUCCESS
T-GRN2.09	TAD	→	DatAna	**4**	TAD GRN E1 published raw inference results to DatAna MQTT GRN E1. TAD GRN E2 published raw inference results to DatAna MQTT GRN E2. TAD GRN F2 published raw inference results to DatAna MQTT GRN F2	24-25/05/2023	SUCCESS

Test ID	Component A	Direction	Component B	I/O ID	Description	Test Date	Result
T-GRN2.10	TAD	←	DatAna	5	TAD GRN E1 consumed raw inference results from CATFlow GRN E1 for Cam-GRN-VA-01 by subscribing to the "CATFlow-V" topic of DatAna MQTT GRN E1. TAD GRN E2 consumed raw inference results from CATFlow GRN E2 for Cam-GRN-VA-02 by subscribing to the "CATFlow-V" topic of DatAna MQTT GRN E2. TAD GRN F2 consumed raw inference results from CATFlow GRN F2 for Cam-GRN-VA-03 by subscribing to the "CATFlow-V" topic of DatAna MQTT GRN F2.	24-25/05/2023	SUCCESS
T-GRN2.11	RBAD	→	DatAna	4	RBAD GRN2 F2.1, F2.2, F2.3 published raw inference results to DatAna MQTT F2	24-25/05/2023	SUCCESS
T-GRN2.12	RBAD	←	DatAna	5	RBAD GRN2 F2.1 consumed raw inference results from CATFlow GRN E1 for Cam-GRN-VA-01 by subscribing to the "CATFlow-V" topic of DatAna MQTT GRN E1. RBAD GRN2 F2.2 consumed raw inference results from CATFlow GRN E2 for Cam-GRN-VA-02 by subscribing to the "CATFlow-V" topic of DatAna MQTT GRN E2. RBAD GRN2 F2.3 consumed raw inference results from CATFlow GRN F2 for Cam-GRN-VA-03 by subscribing to the "CATFlow-V" topic of DatAna MQTT GRN F2.	24-25/05/2023	SUCCESS
T-GRN2.13	SED	→	DatAna	4	SED GRN2 E1 published raw inference results to DatAna MQTT GRN E1. SED GRN2 E2 published raw inference results to DatAna MQTT GRN E2. SED GRN2 F2 published raw inference results to DatAna MQTT GRN F2	24-25/05/2023	SUCCESS
T-GRN2.14	StreamHandler	↔	SmartViz	9	SmartViz C1 made REST API requests to StreamHandler GRN F2 for AV content related to inference results generated by CATFlow, TAD, SED, RBAD. StreamHandler returned the urls for requested AV content.	30-06-23	SUCCESS
T-GRN2.15	StreamHandler	→	SmartViz	10	SmartViz C1 received the AV files for requested inference results (CATFlow, TAD, SED, RBAD) from StreamHandler GRN F2	30-06-23	SUCCESS
T-GRN2.16	DatAna	→	DFB	7	DFB C1 received inference results (CATFlow, TAD, SED, RBAD) from DatAna C1	24-25/05/2023	SUCCESS
T-GRN2.17	DFB	↔	SmartViz	8	SmartViz C1 received inference results (CATFlow, TAD, SED, RBAD) from DFB Kafka / Elastic Search C1 DFB C1 received inference verification messages (CATFlow, TAD, SED, RBAD) from SmartViz C1	24-25/05/2023	SUCCESS

Test ID	Component A	Direction	Component B	I/O ID	Description	Test Date	Result
GRN3: Traffic Anomalous Events							
T-GRN3.01	AV Registry	→	CATFlow	**2**	CATFlow GRN E1, E2, F2 received Camera objects from AV Registry GRN F2 for ids Cam-GRN-VA-01, Cam-GRN-VA-02, Cam-GRN-VA-03	24-25/05/2023	SUCCESS
T-GRN3.02	AV Registry	→	AVAD	**2**	AVAD GRN3 F2.1, F2.2, F2.3 received Camera objects from AV Registry GRN F2 for ids Cam-GRN-VA-01, Cam-GRN-VA-01-Audio, Cam-GRN-VA-02, Cam-GRN-VA-02-Audio, Cam-GRN-VA-03, Cam-GRN-VA-03-Audio	21-06-23	SUCCESS
T-GRN3.03	AV Registry	→	AT	**2**	AT GRN3 F2.1, F2.2, F2.3 received Camera objects from AV Registry GRN F2 for ids Cam-GRN-VA-01-Audio, Cam-GRN-VA-02-Audio, Cam-GRN-VA-03-Audio	24-25/05/2023	SUCCESS
T-GRN3.04	AV Registry	→	StreamHandler	**2**	StreamHandler GRN F2 received Camera objects from AV Registry GRN F2 for ids Cam-GRN-VA-01, Cam-GRN-VA-01-Audio, Cam-GRN-VA-02, Cam-GRN-VA-02-Audio, Cam-GRN-VA-03, Cam-GRN-VA-03-Audio	24-25/05/2023	SUCCESS
T-GRN3.05	AV Registry	→	SmartViz	**2**	SmartViz C1 received Camera objects from AV Registry GRN F2 for ids Cam-GRN-VA-01, Cam-GRN-VA-01-Audio, Cam-GRN-VA-02, Cam-GRN-VA-02-Audio, Cam-GRN-VA-03, Cam-GRN-VA-03-Audio	24-25/05/2023	SUCCESS
T-GRN3.06	VideoAnony	→	CATFlow	**3**	CATFlow GRN E1, E2, F2 received RTSP stream from VideoAnony GRN E1, E2, F2 for ids Cam-GRN-VA-01, Cam-GRN-VA-02, Cam-GRN-VA-03	24-25/05/2023	SUCCESS
T-GRN3.07	VideoAnony	→	AVAD	**3**	AVAD GRN3 F2.1, F2.2, F2.3 received RTSP stream from VideoAnony GRN E1, E2, F2 for ids Cam-GRN-VA-01, Cam-GRN-VA-01-Audio, Cam-GRN-VA-02, Cam-GRN-VA-02-Audio, Cam-GRN-VA-03, Cam-GRN-VA-03-Audio	24-25/05/2023	SUCCESS
T-GRN3.08	VideoAnony	→	AT	**3**	AT GRN3 F2.1, F2.2, F2.3 received RTSP stream from VideoAnony GRN E1, E2, F2 for ids Cam-GRN-VA-01-Audio, Cam-GRN-VA-02-Audio, Cam-GRN-VA-03-Audio	24-25/05/2023	SUCCESS
T-GRN3.09	VideoAnony	→	StreamHandler	**3**	StreamHandler GRN F2 received RTSP stream from VideoAnony GRN E1, E2, F2 for ids Cam-GRN-VA-01, Cam-GRN-VA-01-Audio, Cam-GRN-VA-02, Cam-GRN-VA-02-Audio, Cam-GRN-VA-03, Cam-GRN-VA-03-Audio	24-25/05/2023	SUCCESS

Test ID	Component A	Direction	Component B	I/O ID	Description	Test Date	Result
T-GRN3.10	VideoAnony	→	SmartViz	3	SmartViz C1 received RTSP stream from VideoAnony GRN E1, E2, F2 for ids Cam-GRN-VA-01, Cam-GRN-VA-01-Audio, Cam-GRN-VA-02, Cam-GRN-VA-02-Audio, Cam-GRN-VA-03, Cam-GRN-VA-03-Audio	24-25/05/2023	SUCCESS
T-GRN3.11	CATFlow	→	DatAna	4	CATFlow GRN E1 published raw inference results to DatAna MQTT GRN E1. CATFlow GRN E2 published raw inference results to DatAna MQTT GRN E2. CATFlow GRN F2 published raw inference results to DatAna MQTT GRN F2	24-25/05/2023	SUCCESS
T-GRN3.12	TAD	→	DatAna	4	TAD GRN E1 published raw inference results to DatAna MQTT GRN E1. TAD GRN E2 published raw inference results to DatAna MQTT GRN E2. TAD GRN F2 published raw inference results to DatAna MQTT GRN F2	24-25/05/2023	SUCCESS
T-GRN3.13	TAD	←	DatAna	5	TAD GRN E1 consumed raw inference results from CATFlow GRN E1 for Cam-GRN-VA-01 by subscribing to the "CATFlow-V" topic of DatAna MQTT GRN E1. TAD GRN E2 consumed raw inference results from CATFlow GRN E2 for Cam-GRN-VA-02 by subscribing to the "CATFlow-V" topic of DatAna MQTT GRN E2. TAD GRN F2 consumed raw inference results from CATFlow GRN F2 for Cam-GRN-VA-03 by subscribing to the "CATFlow-V" topic of DatAna MQTT GRN F2.	24-25/05/2023	SUCCESS
T-GRN3.14	AVAD	→	DatAna	4	AVAD GRN3 F2.1, F2.2, F2.3 published raw inference results to DatAna MQTT F2	24-25/05/2023	SUCCESS
T-GRN3.15	AT	→	DatAna	4	AT GRN3 F2.1, F2.2, F2.3 published raw inference results to DatAna MQTT F2	24-25/05/2023	SUCCESS
T-GRN3.16	StreamHandler	←→	SmartViz	9	SmartViz C1 made REST API requests to StreamHandler GRN F2 for AV content related to inference results generated by CATFlow, TAD, AVAD, AT. StreamHandler returned the urls for requested AV content.	30-06-23	SUCCESS
T-GRN3.17	StreamHandler	→	SmartViz	10	SmartViz C1 received the AV files for requested inference results (CATFlow, TAD, AVAD, AT) from StreamHandler GRN F2	30-06-23	SUCCESS
T-GRN3.18	DatAna	→	DFB	7	DFB C1 received inference results (CATFlow, TAD, AVAD, AT) from DatAna C1	24-25/05/2023	SUCCESS

Test ID	Component A	Direction	Component B	I/O ID	Description	Test Date	Result
T-GRN3.19	DFB	↔	SmartViz	**8**	SmartViz C1 received inference results (CATFlow, TAD, AVAD, AT) from DFB Kafka / Elastic Search C1 DFB C1 received inference verification messages (CATFlow, TAD, AVAD, AT) from SmartViz C1	24-25/05/2023	SUCCESS
GRN4: Trajectories							
T-GRN4.01	AV Registry	→	CATFlow	**2**	CATFlow GRN E1, E2, F2 received Camera objects from AV Registry GRN F2 for ids Cam-GRN-VA-01, Cam-GRN-VA-02, Cam-GRN-VA-03	24-25/05/2023	SUCCESS
T-GRN4.02	AV Registry	→	AVCC	**2**	AVCC GRN4 C1 received Camera objects from AV Registry GRN F2 for ids Cam-GRN-VA-01, Cam-GRN-VA-01-Audio	21-06-23	SUCCESS
T-GRN4.03	AV Registry	→	SED	**2**	SED GRN4 C1.1, C1.2, C1.3 received Camera objects from AV Registry GRN F2 for ids Cam-GRN-VA-01-Audio, Cam-GRN-VA-02-Audio, Cam-GRN-VA-03-Audio	24-25/05/2023	SUCCESS
T-GRN4.04	AV Registry	→	SmartViz	**2**	SmartViz C1 received Camera objects from AV Registry GRN F2 for ids Cam-GRN-VA-01, Cam-GRN-VA-01-Audio, Cam-GRN-VA-02, Cam-GRN-VA-02-Audio, Cam-GRN-VA-03, Cam-GRN-VA-03-Audio	24-25/05/2023	SUCCESS
T-GRN4.05	VideoAnony	→	CATFlow	**3**	CATFlow GRN E1, E2, F2 received RTSP stream from VideoAnony GRN E1, E2, F2 for ids Cam-GRN-VA-01, Cam-GRN-VA-02, Cam-GRN-VA-03	24-25/05/2023	SUCCESS
T-GRN4.06	VideoAnony	→	AVCC	**3**	AVCC GRN4 C1 received RTSP stream from VideoAnony GRN E1	24-25/05/2023	SUCCESS
T-GRN4.07	VideoAnony	→	SED	**3**	SED GRN C1.1, C1.2, C1.3 received RTSP stream from VideoAnony GRN E1, F1.1, F1.2	24-25/05/2023	SUCCESS
T-GRN4.08	VideoAnony	→	SmartViz	**3**	SmartViz C1 received RTSP stream from VideoAnony GRN E1, F1.1, F1.2	24-25/05/2023	SUCCESS
T-GRN4.09	CATFlow	→	DatAna	**4**	CATFlow GRN E1 published raw inference results to DatAna MQTT GRN E1. CATFlow GRN F1.1, F1.2 published raw inference results to DatAna MQTT GRN F1	24-25/05/2023	SUCCESS
T-GRN4.10	TAD	→	DatAna	**4**	TAD GRN E1 published raw inference results to DatAna MQTT GRN E1. TAD GRN E2 published raw inference results to DatAna MQTT GRN E2. TAD GRN F2 published raw inference results to DatAna MQTT GRN F2	24-25/05/2023	SUCCESS

Test ID	Component A	Direction	Component B	I/O ID	Description	Test Date	Result
T-GRN4.11	TAD	←	DatAna	5	TAD GRN E1 consumed raw inference results from CATFlow GRN E1 for Cam-GRN-VA-01 by subscribing to the "CATFlow-V" topic of DatAna MQTT GRN E1. TAD GRN E2 consumed raw inference results from CATFlow GRN E2 for Cam-GRN-VA-02 by subscribing to the "CATFlow-V" topic of DatAna MQTT GRN E2. TAD GRN F2 consumed raw inference results from CATFlow GRN F2 for Cam-GRN-VA-03 by subscribing to the "CATFlow-V" topic of DatAna MQTT GRN F2.	24-25/05/2023	SUCCESS
T-GRN4.12	AVCC	→	DatAna	4	AVCC GRN4 C1 published raw inference results to DatAna MQTT C1	24-25/05/2023	SUCCESS
T-GRN4.13	SED	→	DatAna	4	SED GRN4 C1.1, C1.2, C1.3 published raw inference results to DatAna MQTT C1	24-25/05/2023	SUCCESS
T-GRN4.14	DatAna	→	DFB	7	DFB C1 received inference results (CATFlow, TAD, AVCC, SED) from DatAna C1	24-25/05/2023	SUCCESS
T-GRN4.15	DFB	↔	SmartViz	8	SmartViz C1 received inference results (CATFlow, TAD, AVCC, SED) from DFB Kafka / Elastic Search C1 DFB C1 received inference verification messages (CATFlow, TAD, AVCC, SED) from SmartViz C1	24-25/05/2023	SUCCESS (AVCC heatmaps contain errors due to corrupted video frames)
MT1: Crowd Monitoring							
T-MT1.01	AV Registry	→	CATFlow	2	CATFlow MT1 F2.1, F2.2 received Camera objects from AV Registry MT F2 for ids Cam-MT1-VA-01, Cam-MT1-VA-02	24-25/05/2023	SUCCESS
T-MT1.02	AV Registry	→	ViAD	2	thank you very much.	30-06-23	SUCCESS
T-MT1.03	AV Registry	→	VCC	2	VCC MT1 C1.1,C1.2 received Camera objects from AV Registry MT F2 for ids Cam-MT1-VA-01, Cam-MT1-VA-02	30-06-23	SUCCESS
T-MT1.04	AV Registry	→	StreamHandler	2	StreamHandler MT F2 received Camera objects from AV Registry MT F2 for ids Cam-MT1-VA-01, Cam-MT1-VA-02	24-25/05/2023	SUCCESS
T-MT1.05	AV Registry	→	SmartViz	2	SmartViz C1 received Camera objects from AV Registry MT F2 for ids Cam-MT1-VA-01, Cam-MT1-VA-02	24-25/05/2023	SUCCESS
T-MT1.06	RTSP Proxy	→	CATFlow	3	CATFlow MT1 F2.1, F2.2 received RTSP streams from RTSP Proxy MT F2	24-25/05/2023	SUCCESS
T-MT1.07	RTSP Proxy	→	ViAD	3	ViAD MT1 C1.1, C1.2 received RTSP streams from RTSP Proxy MT F2	24-25/05/2023	SUCCESS
T-MT1.08	RTSP Proxy	→	VCC	3	VCC MT1 C1.1, C1.2 received RTSP streams from RTSP Proxy MT F2	24-25/05/2023	SUCCESS

Test ID	Component A	Direction	Component B	I/O ID	Description	Test Date	Result
T-MT1.09	RTSP Proxy	→	StreamHandler	3	StreamHandler MT F2 received RTSP streams from RTSP Proxy MT F2	24-25/05/2023	SUCCESS
T-MT1.10	RTSP Proxy	→	SmartViz	3	SmartViz C1 received RTSP streams from VideoAnony RTSP Proxy MT F2	30-06-23	SUCCESS
T-MT1.11	CATFlow	→	DatAna	4	CATFlow MT1 F2.1, F2.2 published raw inference results to DatAna MQTT MT F2.	24-25/05/2023	SUCCESS
T-MT1.12	ViAD	→	DatAna	4	ViAD MT1 C1.1, C1.2 published raw inference results to DatAna MQTT C1.	24-25/05/2023	SUCCESS
T-MT1.13	VCC	→	DatAna	4	VCC MT1 C1.1, C1.2 published raw inference results to DatAna MQTT C1.	24-25/05/2023	SUCCESS
T-MT1.14	StreamHandler	↔	SmartViz	9	SmartViz C1 made REST API requests to StreamHandler MT F2 for inference results generated by CATFlow, ViAD, VCC. StreamHandler returned the urls for requested AV content.	30-06-23	SUCCESS
T-MT1.15	StreamHandler	→	SmartViz	10	SmartViz C1 received the AV files for requested inference results (CATFlow, ViAD, VCC) from StreamHandler MT F2	30-06-23	SUCCESS
T-MT1.16	DatAna	→	DFB	7	DFB C1 received inference results (CATFlow, ViAD, VCC) from DatAna C1	24-25/05/2023	SUCCESS
T-MT1.17	DFB	↔	SmartViz	8	SmartViz C1 received inference results (CATFlow, ViAD, VCC) from DFB Kafka / Elastic Search C1 DFB C1 received inference verification messages (CATFlow, ViAD, VCC) from SmartViz C1	24-25/05/2023	SUCCESS
MT2: Detecting criminal/anti-social behaviours							
T-MT2.01	AV Registry	→	AAC	2	AAC MT2 F2 received Camera objects from AV Registry MT F2 for id Cam-MT2-AA-01	24-25/05/2023	SUCCESS
T-MT2.02	AV Registry	→	AVAD	2	AVAD MT2 C1.1, C1.2 received Camera objects from AV Registry MT F2 for ids Cam-MT2-VA-01, Cam-MT2-AA-01, Cam-MT2-VA-02, Cam-MT2-AA-02	21-06-23	SUCCESS
T-MT2.03	AV Registry	→	SED	2	SED MT2 C1.1, C1.2 received Camera objects from AV Registry MT F2 for ids Cam-MT2-AA-01, Cam-MT2-AA-02	24-25/05/2023	SUCCESS
T-MT2.04	AV Registry	→	AT	2	AT MT2 C1.1, C1.2 received Camera objects from AV Registry MT F2 for ids Cam-MT2-AA-01, Cam-MT2-AA-02	24-25/05/2023	SUCCESS
T-MT2.05	AV Registry	→	StreamHandler	2	StreamHandler MT F2 received Camera objects from AV Registry MT F2 for ids Cam-MT2-VA-01, Cam-MT2-AA-01, Cam-MT2-VA-02, Cam-MT2-AA-02	24-25/05/2023	SUCCESS

Test ID	Component A	Direction	Component B	I/O ID	Description	Test Date	Result
T-MT2.06	AV Registry	→	SmartViz	2	SmartViz C1 received Camera objects from AV Registry MT F2 for ids Cam-MT2-VA-01, Cam-MT2-AA-01, Cam-MT2-VA-02, Cam-MT2-AA-02	24-25/05/2023	SUCCESS
T-MT2.07	MEMS	→	AudioAnony - VAD	1	AudioAnony-VAD MT2 E1, E2 consumed the raw audio stream from the MEMS microphone MT2 E1, E2	24-25/05/2023	SUCCESS
T-MT2.08	RTSP Proxy	→	AAC	3	AAC MT2 F2 received RTSP stream from RTSP Proxy MT F2	24-25/05/2023	SUCCESS
T-MT2.09	RTSP Proxy	→	AVAD	3	AVAD MT2 C1.1, C1.2 received RTSP stream from RTSP Proxy MT F2	24-25/05/2023	SUCCESS
T-MT2.10	RTSP Proxy	→	SED	3	SED MT2 C1.1, C1.2 received RTSP stream from RTSP Proxy MT F2	24-25/05/2023	SUCCESS
T-MT2.11	RTSP Proxy	→	AT	3	AT MT2 C1.1, C1.2 received RTSP stream from RTSP Proxy MT F2	24-25/05/2023	SUCCESS
T-MT2.12	RTSP Proxy	→	StreamHandler	3	StreamHandler C1 received RTSP stream from RTSP Proxy MT F2	24-25/05/2023	SUCCESS
T-MT2.13	RTSP Proxy	→	SmartViz	3	SmartViz C1 received RTSP stream from RTSP Proxy MT F2	30-06-23	SUCCESS
T-MT2.14	VAD	→	DatAna	4	VAD MT2 E1, E2 published raw inference results to DatAna MQTT MT F2.	24-25/05/2023	SUCCESS
T-MT2.15	AAC	→	DatAna	4	AAC MT2 F2 published raw inference results to DatAna MQTT F2.	24-25/05/2023	SUCCESS
T-MT2.16	AVAD	→	DatAna	4	AVAD MT2 C1.1, C1.2 published raw inference results to DatAna MQTT C1.	24-25/05/2023	SUCCESS
T-MT2.17	SED	→	DatAna	4	SED MT2 C1.1, C1.2 published raw inference results to DatAna MQTT C1.	24-25/05/2023	SUCCESS
T-MT2.18	AT	→	DatAna	4	AT MT2 C1.1, C1.2 published raw inference results to DatAna MQTT C1.	24-25/05/2023	SUCCESS
T-MT2.19	GPURegex	→	DatAna	4	GPURegex MT2 F2 published raw inference results to DatAna MQTT F2.	21-06-23	SUCCESS
T-MT2.20	GPURegex	←	DatAna	5	GPURegex MT2 F2 consumed raw inference results from AAC MT2 F2 for Cam-MT2-AA-01 by subscribing to the "GPURegex" topic of DatAna MQTT MT F2	24-25/05/2023	SUCCESS
T-MT2.21	StreamHandler	↔	SmartViz	9	SmartViz C1 made REST API requests to StreamHandler MT F2 for inference results generated by VAD, AAC, AVAD, SED, AT. StreamHandler returned the urls for requested AV content.	30-06-23	SUCCESS
T-MT2.22	StreamHandler	→	SmartViz	10	SmartViz C1 received the AV files for requested inference results (AAC, AVAD, SED, AT) from StreamHandler MT F2	30-06-23	SUCCESS

Test ID	Component A	Direction	Component B	I/O ID	Description	Test Date	Result
T-MT2.23	DatAna	→	DFB	**7**	DFB C1 received inference results (VAD, AAC, AVAD, SED, AT) from DatAna C1	24-25/05/2023	SUCCESS
T-MT2.24	DFB	↔	SmartViz	**8**	SmartViz C1 received inference results (VAD, AAC, AVAD, SED, AT) from DFB Kafka / Elastic Search C1 DFB C1 received inference verification messages (VAD, AAC, AVAD, SED, AT) from SmartViz C1	21-06-23	SUCCESS
MT3: Parking Lot							
T-MT3.01	AV Registry	→	AVAD	**2**	AVAD MT3 C1 received Camera objects from AV Registry MT F2 for ids Cam-MT3-AA-01, Cam-MT3-VA-01	30-06-23	SUCCESS
T-MT3.02	AV Registry	→	SED	**2**	SED MT3 C1 received Camera objects from AV Registry MT F2 for ids Cam-MT3-AA-01	24-25/05/2023	SUCCESS
T-MT3.03	AV Registry	→	AT	**2**	AT MT3 C1 received Camera objects from AV Registry MT F2 for ids Cam-MT3-AA-01	24-25/05/2023	SUCCESS
T-MT3.04	AV Registry	→	StreamHandler	**2**	StreamHandler MT F2 received Camera objects from AV Registry MT F2 for ids Cam-MT3-AA-01, Cam-MT3-VA-01	24-25/05/2023	SUCCESS
T-MT3.05	AV Registry	→	SmartViz	**2**	SmartViz C1 received Camera objects from AV Registry MT F2 for ids Cam-MT3-AA-01, Cam-MT3-VA-01	24-25/05/2023	SUCCESS
T-MT3.06	MEMS	→	AudioAnony - VAD	**1**	AudioAnony-VAD MT E1 consumed the raw audio stream from the MEMS microphone MT E1	24-25/05/2023	SUCCESS
T-MT3.07	RTSP Proxy	→	StreamHandler	**3**	StreamHandler MT F2 received RTSP stream from RTSP Proxy MT F2	24-25/05/2023	SUCCESS
T-MT3.08	RTSP Proxy	→	SmartViz	**3**	SmartViz C1 received RTSP stream from RTSP Proxy MT F2	30-06-23	SUCCESS
T-MT3.09	RTSP Proxy	→	AVAD	**3**	AVAD MT3 C1 received RTSP stream from RTSP Proxy MT F2	21-06-23	SUCCESS
T-MT3.10	RTSP Proxy	→	SED	**3**	SED MT3 C1 received RTSP stream from RTSP Proxy MT F2	24-25/05/2023	SUCCESS
T-MT3.11	RTSP Proxy	→	AT	**3**	AT MT3 C1 received RTSP stream from RTSP Proxy MT F2	24-25/05/2023	SUCCESS
T-MT3.12	VAD	→	DatAna	**4**	VAD MT3 E1 published raw inference results to DatAna MQTT MT F2.	24-25/05/2023	SUCCESS
T-MT3.13	AVAD	→	DatAna	**4**	AVAD MT3 C1 published raw inference results to DatAna MQTT C1.	21-06-23	SUCCESS
T-MT3.14	SED	→	DatAna	**4**	SED MT3 C1 published raw inference results to DatAna MQTT C1.	24-25/05/2023	SUCCESS
T-MT3.15	AT	→	DatAna	**4**	AT MT3 C1 published raw inference results to DatAna MQTT C1.	24-25/05/2023	SUCCESS

Test ID	Component A	Direction	Component B	I/O ID	Description	Test Date	Result
T-MT3.16	StreamHandler	↔	SmartViz	9	SmartViz C1 made REST API requests to StreamHandler MT F2 for inference results generated by VAD, AVAD, SED, AT. StreamHandler returned the urls for requested AV content.	30-06-23	SUCCESS
T-MT3.17	StreamHandler	→	SmartViz	10	SmartViz C1 received the AV files for requested inference results (AVAD, SED, AT) from StreamHandler MT F2	30-06-23	SUCCESS
T-MT3.18	DatAna	→	DFB	7	DFB C1 received inference results (VAD, AVAD, SED, AT) from DatAna C1	21-06-23	SUCCESS
T-MT3.19	DFB	↔	SmartViz	8	SmartViz C1 received inference results (VAD, AVAD, SED, AT) from DFB Kafka / Elastic Search C1 DFB C1 received inference verification messages (VAD, AVAD, SED, AT) from SmartViz C1	21-06-23	SUCCESS
MT4: Analysis of a specific area							
T-MT4.01	AV Registry	→	CATFlow	2	CATFLow MT4 F2.1, F2.2 received Camera objects from AV Registry MT F2 for id Cam-MT4-VA-01, Cam-MT4-VA-02	24-25/05/2023	SUCCESS
T-MT4.02	AV Registry	→	AVAD	2	AVAD MT4 C1 received Camera objects from AV Registry MT F2 for ids Cam-MT4-VA-01, Cam-MT4-AA-01	30-06-23	SUCCESS
T-MT4.03	AV Registry	→	SED	2	SED MT4 C1 received Camera objects from AV Registry MT F2 for id Cam-MT4-AA-01	24-25/05/2023	SUCCESS
T-MT4.04	AV Registry	→	StreamHandler	2	StreamHandler MT F2 received Camera objects from AV Registry MT F2 for ids Cam-MT4-VA-01, Cam-MT4-AA-01, Cam-MT4-VA-02	24-25/05/2023	SUCCESS
T-MT4.05	AV Registry	→	SmartViz	2	SmartViz C1 received Camera objects from AV Registry MT F2 for ids Cam-MT4-VA-01, Cam-MT4-AA-01, Cam-MT4-VA-02	24-25/05/2023	SUCCESS
T-MT4.06	MEMS	→	AudioAnony - VAD	1	AudioAnony-VAD MT4 E1 consumed the raw audio stream from the MEMS microphone MT4 E1	24-25/05/2023	SUCCESS
T-MT4.07	RTSP Proxy	→	CATFlow	3	CATFlow MT4 F2.1, F2.2 received RTSP stream from RTSP Proxy MT F2	24-25/05/2023	SUCCESS
T-MT4.08	RTSP Proxy	→	AVAD	3	AVAD MT4 C1 received RTSP stream from RTSP Proxy MT F2	24-25/05/2023	SUCCESS
T-MT4.09	RTSP Proxy	→	SED	3	SED MT4 C1 received RTSP stream from RTSP Proxy MT F2	24-25/05/2023	SUCCESS
T-MT4.10	RTSP Proxy	→	StreamHandler	3	StreamHandler C1 received RTSP stream from RTSP Proxy MT F2	24-25/05/2023	SUCCESS
T-MT4.11	RTSP Proxy	→	SmartViz	3	SmartViz C1 received RTSP stream from RTSP Proxy MT F2	30-06-23	SUCCESS

Test ID	Component A	Direction	Component B	I/O ID	Description	Test Date	Result
T-MT4.12	VAD	→	DatAna	**4**	VAD MT4 E1 published raw inference results to DatAna MQTT MT F2.	24-25/05/2023	SUCCESS
T-MT4.13	CATFlow	→	DatAna	**4**	CATFlow MT4 F2.1, F2.2 published raw inference results to DatAna MQTT F2.	24-25/05/2023	SUCCESS
T-MT4.14	AVAD	→	DatAna	**4**	AVAD MT4 C1 published raw inference results to DatAna MQTT C1.	24-25/05/2023	SUCCESS
T-MT4.15	SED	→	DatAna	**4**	SED MT4 C1 published raw inference results to DatAna MQTT C1.	24-25/05/2023	SUCCESS
T-MT4.16	StreamHandler	↔	SmartViz	**9**	SmartViz C1 made REST API requests to StreamHandler MT F2 for inference results generated by VAD, CATFLow, AVAD, SED. StreamHandler returned the urls for requested AV content.	30-06-23	SUCCESS
T-MT4.17	StreamHandler	→	SmartViz	**10**	SmartViz C1 received the AV files for requested inference results (CATFlow, AVAD, SED) from StreamHandler MT F2	30-06-23	SUCCESS
T-MT4.18	DatAna	→	DFB	**7**	DFB C1 received inference results (VAD, CATFlow, AVAD, SED) from DatAna C1	24-25/05/2023	SUCCESS
T-MT4.19	DFB	↔	SmartViz	**8**	SmartViz C1 received inference results (VAD, CATFlow, AVAD, SED) from DFB Kafka / Elastic Search C1 DFB C1 received inference verification messages (VAD, CATFlow, AVAD, SED) from SmartViz C1	24-25/05/2023	SUCCESS
UNS1: Drone Crowd Classification							
T-UNS1.01	AV Registry	→	VCC	**2**	VCC UNS1 F1 received Camera objects from AV Registry UNS F1 for id Cam-Cam-UNS-VA-01	15-06-23	SUCCESS
T-UNS1.02	AV Registry	→	StreamHandler	**2**	StreamHandler UNS F1 received Camera objects from AV Registry UNS F1 for ids Cam-UNS-VA-01, Cam-UNS-AA-01	24-25/05/2023	SUCCESS
T-UNS1.03	AV Registry	→	SmartViz	**2**	SmartViz C1 received Camera objects from AV Registry UNS F1 for ids Cam-UNS-VA-01, Cam-UNS-AA-01	24-25/05/2023	SUCCESS
T-UNS1.04	MEMS	→	AudioAnony - VAD	**1**	AudioAnony-VAD UNS E2 consumed the raw audio stream from the MEMS microphone UNS E2	24-25/05/2023	SUCCESS
T-UNS1.05	EdgeSec TEE	→	VideoAnony	**N/A**	VideoAnony requests and receives information from EdgeSec TEE over a REST protocol	15-06-23	SUCCESS
T-UNS1.06	VideoAnony	→	VCC	**3**	VCC UNS1 F1 received RTSP stream from VideoAnony UNS E1	24-25/05/2023	SUCCESS

Test ID	Component A	Direction	Component B	I/O ID	Description	Test Date	Result
T-UNS1.07	VideoAnony	→	StreamHandler	**3**	StreamHandler UNS F1 received RTSP stream from VideoAnony UNS E1	24-25/05/2023	SUCCESS
T-UNS1.08	VideoAnony	→	SmartViz	**3**	SmartViz C1 received RTSP stream from VideoAnony UNS E1	24-25/05/2023	SUCCESS
T-UNS1.09	AudioAnony - VAD	→	StreamHandler	**3**	StreamHandler UNS F1 received RTSP stream from AudioAnony-VAD UNS E2	24-25/05/2023	SUCCESS
T-UNS1.10	AudioAnony - VAD	→	SmartViz	**3**	SmartViz C1 received RTSP stream from AudioAnony-VAD UNS E2	30-06-23	SUCCESS
T-UNS1.11	VAD	→	DatAna	**4**	VAD UNS E2 published raw inference results to DatAna MQTT UNS E2.	24-25/05/2023	SUCCESS
T-UNS1.12	VCC	→	DatAna	**4**	VCC UNS1 F1 published raw inference results to DatAna MQTT UNS F1.	24-25/05/2023	SUCCESS
T-UNS1.13	StreamHandler	↔	SmartViz	**9**	SmartViz C1 made REST API requests to StreamHandler UNS F1 for inference results generated by VCC. StreamHandler returned the urls for requested AV content.	15-06-23	SUCCESS
T-UNS1.14	StreamHandler	→	SmartViz	**10**	SmartViz C1 received the AV files for requested inference results (VAD, VCC) from StreamHandler UNS F1	15-06-23	SUCCESS
T-UNS1.15	DatAna	→	DFB	**7**	DFB C1 received inference results (VAD, VCC) from DatAna C1	24-25/05/2023	SUCCESS
T-UNS1.16	DFB	↔	SmartViz	**8**	SmartViz C1 received inference results (VAD, VCC) from DFB Kafka / Elastic Search C1 DFB C1 received inference verification messages (VAD, VCC) from SmartViz C1	24-25/05/2023	SUCCESS
UNS2: Sound Localisation							
T-UNS2.01	AV Registry	→	SELD	**2**	SELD UNS2 F1 received Camera objects from AV Registry UNS F1 for id Cam-UNS-MEMS-02	24-25/05/2023	SUCCESS
T-UNS2.02	AV Registry	→	StreamHandler	**2**	StreamHandler UNS F1 received Camera objects from AV Registry UNS F1 for ids Cam-UNS-AA-02	24-25/05/2023	SUCCESS
T-UNS2.03	AV Registry	→	SmartViz	**2**	SmartViz C1 received Camera objects from AV Registry UNS F1 for id Cam-UNS-AA-02	24-25/05/2023	SUCCESS
T-UNS2.04	MEMS	→	AudioAnony - VAD	**1**	AudioAnony-VAD UNS E3 consumed the raw audio stream from the MEMS microphone UNS E3	07-06-23	SUCCESS
T-UNS2.05	MEMS	→	SELD	**3**	SELD UNS F1 received RTSP stream from RTSPProxy UNS E3	07-06-23	SUCCESS
T-UNS2.06	AudioAnony - VAD	→	StreamHandler	**3**	StreamHandler UNS F1 received RTSP stream from AudioAnony-VAD UNS E3	07-06-23	SUCCESS

Test ID	Component A	Direction	Component B	I/O ID	Description	Test Date	Result
T-UNS2.07	AudioAnony - VAD	→	SmartViz	**3**	SmartViz C1 received RTSP stream from AudioAnony-VAD UNS E3	30-06-23	SUCCESS
T-UNS2.08	VAD	→	DatAna	**4**	VAD UNS E3 published raw inference results to DatAna MQTT UNS F1.	30-06-23	FAIL (resolution in progress)
T-UNS2.09	SELD	→	DatAna	**4**	SELD UNS F1 published raw inference results to DatAna MQTT UNS F1.	15-06-23	SUCCESS
T-UNS2.10	StreamHandler	←→	SmartViz	**9**	SmartViz C1 made REST API requests to StreamHandler UNS F1 for inference results generated by SELD. StreamHandler returned the urls for requested AV content.	30-06-23	SUCCESS
T-UNS2.11	StreamHandler	→	SmartViz	**10**	SmartViz C1 received the AV files for requested inference results (VAD, SELD) from StreamHandler UNS F1	30-06-23	SUCCESS
T-UNS2.12	DatAna	→	DFB	**7**	DFB C1 received inference results (VAD, SELD) from DatAna C1	15-06-23	SUCCESS
T-UNS2.13	DFB	←→	SmartViz	**8**	SmartViz C1 received inference results (VAD, SELD) from DFB Kafka / Elastic Search C1 DFB C1 received inference verification messages (VAD, SELD) from SmartViz C1	15-06-23	SUCCESS

Appendix B: R2 Data Model specifications

“Camera” SDM-compliant Data Model for AV Sources

Entity: Camera

Global description:

A Data Model for all AV sources in the MARVEL framework (including generated anonymized AV content). Based on the Smart Data Models (SDM) standard

Based on the Smart Data Models (SDM) specification.

Original Data Model by SDM initiative

Open License

List of properties

Properties by SDM

- `address`: The mailing address
- `alternateName`: An alternative name for this item
- `areaServed`: The geographic area where a service or offered item is provided
- `cameraName`: Name of the camera corresponding to this observation.
- `cameraOrientation`: Orientation information for the camera corresponding to this observation
- `cameraType`: Type of the camera corresponding to this observation. Enum:'FIXED, PTZ, DOME, DAY/NIGHT, C-MOUNT, BULLET'.
- `cameraUsage`: Purpose of the camera corresponding to this observation. ENUM: [SURVEILLANCE, RLVD, ANPR/LPR].
- `dataProvider`: A sequence of characters identifying the provider of the harmonised data entity.
- `dateCreated`: Entity creation timestamp. This will usually be allocated by the storage platform.
- `dateModified`: Timestamp of the last modification of the entity. This will usually be allocated by the storage platform.
- `description`: A description of this item
- `endDateTime`: Reported end time corresponding to this observation.
- `imageSnapshot`: Camera feed snapshot download link for the camera corresponding to this observation
- `location`: Geojson reference to the item. It can be Point, LineString, Polygon, MultiPoint, MultiLineString or MultiPolygon
- `mediaURL`: URL providing further information of any image(s) or media of the complaint or place.
- `name`: The name of this item.
- `seeAlso`: list of uri pointing to additional resources about the item
- `source`: A sequence of characters giving the original source of the entity data as a URL. Recommended to be the fully qualified domain name of the source provider, or the URL to the source object.
- `startDateTime`: Reported start time corresponding to this observation.
- `streamName`: Name of the video stream from the camera corresponding to this observation
- `streamURL`: URL providing video streaming information for the camera corresponding to this observation

Properties by SDM modified for MARVEL

- `cameraNum`: Camera number corresponding to this observation (##).
- `owner`: A List containing a JSON encoded sequence of characters referencing the unique Ids of the owner(s). In the case of MARVEL, we can use the pilot name and use case here. ENUM: [GRN3, GRN4, MT1, MT3, UNS1].
- `id`: Unique identifier of the entity. IDs for AV sources will be hard-coded manually in a configuration file or automatically assigned by the AV registry component. Recommended format: "Cam-" + owner + "-" AVSourceType + "-" + cameraNum(##).
- `type`: The type of the entity. In this case "Camera".

Additional properties by MARVEL

- `AVSourceType`: Indicates whether the AV source is a CCTV camera, a MEMS microphone or an anonymisation service (VideoAnony or AudioAnony or both). ENUM: [CCTV, MEMS, VA, AA, VAAA]
- `contentType`: Indicates whether the source transmits audio, video or both. ENUM: [A, V, AV].
- `cameraManufacturer`: The name of the manufacturer producing the camera.
- `cameraModel`: The name of the camera model.
- `videoAnonymized`: Indicates whether the specific AV source provides anonymised video content. (boolean type)
- `audioAnonymized`: Indicates whether the specific AV source provides anonymised audio content. (boolean type)
- `originalAVSourceId`: Points to the id of the original AV source in case the current source is anonymised. In case this is the original AV source, then the camera's own id is used.
- `videoResolutionWidth`: The width of the frame in pixels for. Applicable for sources that contain Video content.
- `videoResolutionHeight`: The height of the frame in pixels. Applicable for sources that contain Video content.
- `videoFramerate`: The video framerate in fps. Applicable for sources that contain Video content.
- `videoCodec`: The codec that was used to encode the video stream. Applicable for sources that contain Video content. ENUM: [H264, H265, MPEG4].
- `videoBitrate`: The bitrate of the video in the AV stream (in Kbps).
- `audioSamplingRate`: The sampling rate of the audio. Applicable for sources that contain Audio content.
- `audioBitDepth`: The bit-depth of the audio. ENUM: [8-bit, 16-bit, 24-bit, 32-bit]
- `audioCodec`: The codec that was used to encode the audio stream. Applicable for sources that contain Audio content. ENUM: [MP3, WAV, PCM, AAC, AIFF, OGG].
- `audioBitrate`: The bitrate of the audio in the AV stream (in Kbps).
- `audioGain`: The gain of the audio stream (float value)
- `audioAGC`: Indicates whether Automatic Gain Control is applied (boolean value).
- `audioChannels`: The number of channels in the audio stream (integer value).
- `AVContainer`: The container used to encapsulate the AV data. ENUM: [MP4, MOV, AVI, WAV, MP3, AAC].

Required properties by SDM

- `id`
- `type`

Required properties by MARVEL

- `streamURL`
- `owner`
- `contentType`
- `cameraManufacturer`
- `cameraModel`
- `anonymized`
- `originalAVSourceId`
- `resolutionWidth`
- `resolutionHeight`
- `framerate`
- `samplingRate`
- `videoCodec`
- `audioCodec`
- `AVContainer`

Example payload

```json
{
  "id": "Cam-GRN4-VAAA-03",
  "type": "Camera",
  "AVSourceType": "VAAA",
  "owner": "GRN4",
  "cameraNum": "03",
  "cameraName": "GRN4-VAAA-03",
  "streamName": "GRN_Mgarr_XxxxxxStr_Surv_Fixed_Cam-aca8b8870f61",
  "streamURL": "rtsp://wowzaec2demo.streamlock.net/vod/mp4:BigBuckBunny_115k.mp4",
  "cameraManufacturer": "FBK",
  "cameraModel": "VideoAnony_v05-AudioAnony_v03",
  "location": {
    "type": "Point",
    "coordinates": [
      14.366710,
      35.920308
    ]
  },
  "cameraOrientation": {
    "comments": "Camera facing East",
  },
  "contentType": "AV",
  "videoAnonymized": true,
  "audioAnonymized": true,
  "originalAVSourceId": "Cam-GRN4-CCTV-17",
  "videoResolutionWidth": 1600,
  "videoResolutionHeight": 1200,
  "videoFramerate": 30,
  "videoCodec": "H264",
  "videoBitrate": 20000,
  "audioSamplingRate": 44100,
  "audioBitDepth": "24-bit",
  "audioCodec": "MP3",
  "audioBitrate": 240,
  "audioGain": 1.0,
  "audioAGC": false,
  "audioChannels": 1,
  "AVContainer": "MP4",
}
```

“MediaEvent” SDM-compliant Inference Result Data Model

Entity: MediaEvent

Global description:

A Data Model for events generated by AI components in the MARVEL framework. Based on the Smart Data Models (SDM) standard

Based on the Smart Data Models (SDM) specification.

Original Data Model by SDM initiative

Open License

List of properties

- `address`: The mailing address
- `alternateName`: An alternative name for this item
- `areaServed`: The geographic area where a service or offered item is provided
- `dataProvider`: A sequence of characters identifying the provider of the harmonised data entity.
- `description`: A description of this item
- `eventType`: Type of event that was raised. (ie: PlateDetectionEvent, ColourDetectionEvent, etc.
- `id`: Unique identifier of the entity
- `location`: Geojson reference to the item. It can be Point, LineString, Polygon, MultiPoint, MultiLineString or MultiPolygon
- `mediaSource`: Technical information of the object that raised the event
- `name`: The name of this item.
- `observedEntities`: Array of model Entities created updated or just observed by this event.
- `seeAlso`: list of uri pointing to additional resources about the item
- `source`: A sequence of characters giving the original source of the entity data as a URL. Recommended to be the fully qualified domain name of the source provider, or the URL to the source object.
- `type`: NGSI Entity type. It has to be MediaEvent

Properties by SDM modified for MARVEL

- `data`: Any serializable object that is attached to the event. Eg:plate-number + Attribute type. This is where MARVEL AI components will include their case-specific inference results.
- `detectedBy`: The ID of the device at which the event was detected (the link to the device id of the infrastructure in MARVEL).
- `owner`: A List containing a JSON encoded sequence of characters referencing the unique Ids of the owner(s). In the case of MARVEL, we can use the pilot name and use case here. ENUM: [GRN, GRN1, GRN2, GRN3, GRN4, MT1, MT2 MT3, MT4, UNS1, UNS2].
- `dateCreated`: Entity creation timestamp. Timestamp in which DatAna performs the transformation of the event
- `dateModified`: Timestamp of the last modification of the entity. This will usually be allocated by the storage platform. In MARVEL this field contains the last update by other components (i.e. SmartViz)

Additional properties by MARVEL

- `cameraId`: The id of the Camera entity that produced the AV stream and that was analysed to generate this event. Formatted according to the NGSI-LD standard (i.e. URN identifier).

IDs for AV sources will be hard-coded manually in a configuration file or automatically assigned by the AV registry component and will include a 12-digit hexadecimal identifier.

- `startTime`: The start time of the event. Following the ISO8601 UTC format.
- `endTime`: The end time of the event. Following the ISO8601 UTC format.
- `MLModelId`: The id of the MLModel that generated this event.
- `reviewed`: (boolean) Indicates whether the inference result has been reviewed by the user. Default value false.
- `verified`: (boolean) Indicates whether the inference result is verified by the user or not. Default value false.
- `timestamp`: Entity creation timestamp. In MARVEL, this is the absolute timestamp pointing to the exact time of the video stream to which this element is referring to.
- `dateDetected`: Timestamp in which the inference model processed and detected the entity.
- `dateProcessed`: Timestamp in which the inference model processed and detected the entity. In MARVEL this is the date when the event finalise its process in DatAna.
- `dateStored`: Timestamp in which the entity is persisted. This will usually be allocated by the storage platform. In MARVEL this is done by the DFB when storing the entity in Elastic Search.
- `secondCameraId`: The id of a second Camera entity (audio or video) that produced a AV stream. This is used only for some inference components that need to digest two streams (audio / video) at the same time, and it is complementary and following the same conventions as the cameraId (main stream). Used so far for VCC and AVCC, not mandatory.

Required properties by SDM

- `dateCreated`
- `eventType`
- `id`
- `type`

Required properties by MARVEL

- `cameraId`
- `startTime`
- `endTime`
- `owner`
- `data`
- `MLModelId`
- `detectedBy`
- `dateDetected`
- `dateCreated`
- `reviewed`
- `verified`
- `timestamp` - Either timestamp or startTime/endTime are mandatory

The data field is open in order to be able to contain any information detected by custom filters. For example a filter for trafic plates may just have as data the plate number but a fencing filter might have as data the criticality, the actual coordinates of the violation, and the url of a taken picture or even the BASE64 image

Example payload

```
{
  "id": "mediaEvent_1509702324600",
  "startTime": "",
  "endTime": "",
  "dateCreated": "2022-04-01T10:07:59.618+01:58",
  "dateDetected": "2022-04-01T10:08:24.620+02:00",
  "timestamp": "2022-04-01T10:07:53.614+01:54",
  "owner": "GRN4",
  "cameraId": "Cam-MGARR",
  "MLModelId": "AVCC_v01",
  "detectedBy": "GRN_Fog_Server",
  "alertSource": "MLModel",
  "type": "MediaEvent",
  "eventType": "crowd-detected",
  "reviewed": "false",
  "verified": "false",
  "data": {
    "frame": 0,
    "predicted_count": "18.0",
    "image_paths": "frames/0.jpg",
    "audio_paths": "audio/0.wav",
    "density_paths": null,
    "inference_time": "2.33"
  }
}
```

"Alert" SDM-compliant Inference Result Data Model

Entity: Alert

Global description:

A Data Model for alerts generated by AI components in the MARVEL framework. Based on the Smart Data Models (SDM) standard.

Based on the Smart Data Models (SDM) specification.

Original Data Model by SDM initiative

Open License

List of properties

- `address`: The mailing address
- `alternateName`: An alternative name for this item
- `areaServed`: The geographic area where a service or offered item is provided
- `category`: Category of the Alert. Enum:'traffic, naturalDisaster, weather, environment, health, security, agriculture'
- `dataProvider`: A sequence of characters identifying the provider of the harmonised data entity.
- `description`: A description of this item
- `id`: Unique identifier of the entity
- `location`: Geojson reference to the item. It can be Point, LineString, Polygon, MultiPoint, MultiLineString or MultiPolygon
- `name`: The name of this item.
- `seeAlso`: list of uri pointing to additional resources about the item
- `severity`: Severity of the Alert (informational, low, medium, high, critical)
- `source`: A sequence of characters giving the original source of the entity data as a URL. Recommended to be the fully qualified domain name of the source provider, or the URL to the source object.
- `subCategory`: Describe the sub category of alert. Enum:'trafficJam, carAccident, carWrongDirection, carStopped, pothole, roadClosed, roadWorks, hazardOnRoad, injuredBiker, pedestrianOnRoad, bikerOnRoad, tramApproaching, flood, tsunami, coastalEvent, earthquake, rainfall, highTemperature, lowTemperature, heatWave, coldWave, ice, snow, wind, fog, tornado, tropicalCyclone, hurricane, snow/ice, thunderstorms, fireRisk, avalancheRisk, floodRisk, airPollution, waterPollution, pollenConcentration, asthmaAttack, bumpedPatient, fallenPatient, heartAttack, suspiciousAction, robbery, assault, civilDisorder, buildingFire, forestFire, noxiousWeed, snail, insect, rodent, bacteria, microbe, fungus,mite, virus, nematodes, irrigation, fertilisation
- `type`: NGSI Entity type. It has to be Alert.
- `validFrom`: The start of the validity period for this forecast as a ISO8601 format
- `validTo`: The end of the validity period for this forecast as a ISO8601 format

Properties by FIWARE modified for MARVEL

- `alertSource`: Source of the alert (can ve a user, an application, a model -"MLModel" is the default in MARVEL)
- `data`: Payload containing the data retrieved. This is where MARVEL AI components will include their case-specific inference results.

- `owner`: A List containing a JSON encoded sequence of characters referencing the unique Ids of the owner(s). In the case of MARVEL, we can use the pilot name and use case here. ENUM: [GRN, GRN1, GRN2, GRN3, GRN4, MT1, MT2 MT3, MT4, UNS1, UNS2].
- `dateCreated`: Entity creation timestamp. Timestamp in which DatAna performs the transformation of the event
- `dateModified`: Timestamp of the last modification of the entity. This will usually be allocated by the storage platform. In MARVEL this field contains the last update by other components (i.e. SmartViz)

Additional properties by MARVEL

- `cameraId`: The id of the Camera or mic entity that produced the AV stream and that was analysed to generate this event. Formatted according to the MARVEL camera or microphon id conventions. IDs for AV sources will be hard-coded manually in a configuration file or automatically assigned by the AV registry component and will include a 12-digit hexadecimal identifier.
- `startTime`: The start time of the event. Following the ISO8601 UTC format.
- `endTime`: The end time of the event. Following the ISO8601 UTC format.
- `MLModelId`: The id of the MLModel that generated this event.
- `reviewed`: (boolean) Indicates whether the inference result has been reviewed by the user. Default value false.
- `verified`: (boolean) Indicates whether the inference result is verified by the user or not. Default value false.
- `detectedBy`: Link to the device providing where the alert is detected (the link to the device id of the infrastructure in MARVEL)
- `timestamp`: Entity creation timestamp. In MARVEL, this is the absolute timestamp pointing to the exact time of the video stream to which this element is referring to.
- `dateDetected`: Timestamp in which the inference model processed and detected the entity.
- `dateProcessed`: Timestamp in which the inference model processed and detected the entity. In MARVEL this is the date when the event finalise its process in DatAna.
- `dateStored`: Timestamp in which the entity is persisted. This will usually be allocated by the storage platform. In MARVEL this is done by the DFB when storing the entity in Elastic Search.

Required properties by SDM

- `alertSource`
- `category`
- `id`
- `type`

Required properties by MARVEL

- `cameraId`
- `startTime`
- `endTime`
- `dateCreated`
- `dateDetected`
- `owner`
- `data`
- `MLModelId`
- `detectedBy`

- `reviewed`
- `verified`
- `timestamp` - Either timestamp or startTime/endTime are mandatory

This entity models an alert and could be used to send alerts related to traffic jam, accidents, weather conditions, high level of pollutants and so on. The purpose of the model is to support the generation of notifications for a user or trigger other actions, based on such alerts. An alert is generated by a specific situation. The main features of an alert is that it is not predictable and it is not a recurrent data. That means that an alert could be an accident or a high level of pollutants measure, additionally it could be the fall down of a patient or a car driving in the opposite direction. Some examples of context data are; type of alert (traffic, weather, security, and pollution, etc.), severity, location and so on.

Example payload

```
{
  "id": "Alert_1",
  "startTime": "",
  "endTime": "",
  "dateCreated": "2022-04-01T10:07:59.618+01:58",
  "dateIssued": "2022-04-01T10:08:24.620+02:00",
  "timestamp": "2022-04-01T10:07:53.614+01:54",
  "owner": "GRN4",
  "cameraId": "Cam-MGARR",
  "MLModelId": "AVCC_v01",
  "alertSource": "MLModel",
  "type": "Alert",
  "detectedBy": "GRN_Fog_Server",
  "category": "traffic",
  "subCategory":"trafficJam",
  "description": "The area is getting crowded",
  "severity": "high",
  "reviewed": "false",
  "verified": "false",
  "data": {
    "frame": 0,
    "predicted_count": "18.0",
    "image_paths": "frames/0.jpg",
    "audio_paths": "audio/0.wav",
    "density_paths": null,
    "inference_time": "2.33"
  }
}
```

“Anomaly” SDM-compliant Inference Result Data Model

Entity: Anomaly

Global description:

A Data Model for anomalies detected by AI components in the MARVEL framework. Based on the Smart Data Models (SDM) standard. This entity contains a harmonised description of an anomaly.

Based on the Smart Data Models (SDM) specification.

Original Data Model by SDM initiative

Open License

List of properties

- `address`: The mailing address
- `alternateName`: An alternative name for this item
- `anomalousProperty`: The controlledProperty (of the device) in which the anomaly was detected
- `areaServed`: The geographic area where a service or offered item is provided
- `dataProvider`: A sequence of characters identifying the provider of the harmonised data entity.
- `description`: A description of this item
- `id`: Unique identifier of the entity
- `location`: Geojson reference to the item. It can be Point, LineString, Polygon, MultiPoint, MultiLineString or MultiPolygon
- `name`: The name of this item.
- `seeAlso`: list of uri pointing to additional resources about the item
- `source`: A sequence of characters giving the original source of the entity data as a URL. Recommended to be the fully qualified domain name of the source provider, or the URL to the source object.
- `thresholdBreach`: Description of an observed threshold breach that contributed to detection of an anomaly
- `type`: NGSI-LD Entity Type. It has to be Anomaly

Properties by SDM modified for MARVEL

- `detectedBy`: The ID of the device at which the anomaly was detected (the link to the device id of the infrastructure in MARVEL).
- `owner`: A List containing a JSON encoded sequence of characters referencing the unique Ids of the owner(s). In the case of MARVEL, we can use the pilot name and use case here. ENUM: [GRN, GRN1, GRN2, GRN3, GRN4, MT1, MT2 MT3, MT4, UNS1, UNS2].
- `dateCreated`: Entity creation timestamp. Timestamp in which DatAna performs the transformation of the event
- `dateDetected`: Timestamp in which the inference model processed and detected the entity.
- `dateModified`: Timestamp of the last modification of the entity. This will usually be allocated by the storage platform. In MARVEL this field contains the last update by other components (i.e. SmartViz)

Additional properties by MARVEL

- `cameraId`: The id of the Camera entity that produced the AV stream and that was analysed to generate this event. Formatted according to the NGSI-LD standard (i.e. URN identifier).

IDs for AV sources will be hard-coded manually in a configuration file or automatically assigned by the AV registry component and will include a 12-digit hexadecimal identifier.

- `startTime`: The start time of the event. Following the ISO8601 UTC format.
- `endTime`: The end time of the event. Following the ISO8601 UTC format.
- `data`: Payload containing the data retrieved. This is where MARVEL AI components will include their case-specific inference results.
- `MLModelId`: The id of the MLModel that generated this event.
- `reviewed`: (boolean) Indicates whether the inference result has been reviewed by the user. Default value false.
- `verified`: (boolean) Indicates whether the inference result is verified by the user or not. Default value false.
- `timestamp`: Entity creation timestamp. In MARVEL, this is the absolute timestamp pointing to the exact time of the video stream to which this element is referring to.
- `dateProcessed`: Timestamp in which the inference model processed and detected the entity. In MARVEL this is the date when the event finalise its process in DatAna.
- `dateStored`: Timestamp in which the entity is persisted. This will usually be allocated by the storage platform. In MARVEL this is done by the DFB when storing the entity in Elastic Search.

Required properties by SDM

- `anomalousProperty`
- `dateDetected`
- `id`
- `type`

Required properties by MARVEL

- `cameraId`
- `dateCreated`
- `dateDetected`
- `startTime`
- `endTime`
- `owner`
- `data`
- `MLModelId`
- `deviceSource`
- `reviewed`
- `verified`
- `timestamp` - Either timestamp or startTime/endTime are mandatory

Example payload

```
{
  "id": "Anomaly_1",
  "startTime": "2022-04-01T10:08:24.620+02:00",
  "endTime": "",
  "dateCreated": "2022-04-01T10:07:59.618+01:58",
  "dateDetected": "2022-04-01T10:08:24.620+02:00",
  "timestamp": "2022-04-01T10:07:53.614+01:54",
  "owner": "GRN4",
  "cameraId": "Cam-MGARR",
  "MLModelId": "AVCC_v01",
  "detectedBy": "GRN_Fog_Server"
  "type": "Anomaly",
  "anomalousProperty": "predicted_count",
  "data": {
    "frame": 0,
    "predicted_count": "18.0",
    "image_paths": "frames/0.jpg",
    "audio_paths": "audio/0.wav",
    "density_paths": null,
    "inference_time": "2.33",
  },
  "thresholdBreach": [
    {
      "dateObserved": "2022-04-01T10:08:24.620+02:00",
      "measuredValue": 18.00,
      "thresholdType": "LOWER",
      "thresholdValue": 15
    },
    {
      "dateObserved": "2022-04-02T10:10:25.620+02:11",
      "measuredValue": 18.00,
      "thresholdType": "LOWER",
      "thresholdValue": 15
    }
  ]
}
```

"MLModel" SDM-compliant Data Model for AI model descriptors

Entity: MLModel

Global description:

Data model for compilation of the elements of a machine learning model in the MARVEL framework.

Original Data Model by SDM initiative

Open License

List of properties

Mandatory fields for MARVEL marked with "*".

- `acceptableDataSources*`: Valid type of input data sources for running the Machine Learning Model
- `algorithm*`: The algorithm used by the underlying Machine Learning model (e.g. linear regression, k-means, SVM, MLP,...)
- `alternateName`: An alternative name for this item
- `dataProvider`: A sequence of characters identifying the provider of the harmonised data entity.
- `dateCreated*`: Entity creation timestamp. This will usually be allocated by the storage platform.
- `dateModified*`: Timestamp of the last modification of the entity. This will usually be allocated by the storage platform.
- `description*`: A description of this item
- `dockerImage`: Docker image containing the Machine Learning model
- `id*`: Unique identifier of the entity
- `inputAttributes*`: Comma-separated list of attributes names (that should have a given type by definition).
- `mlFramework*`: The Machine Learning framework that has been used to prepare the model (e.g., scikit-learn, H2O, Spark MLib, etc)
- `name*`: The name of this item.
- `outputAttributes*`: Comma-separated list of attributes names used to publish the results.
- `outputDataTypes*`: Type of output data produced by the Machine Learning Model
- `owner`: A List containing a JSON encoded sequence of characters referencing the unique Ids of the owner(s)
- `refMLProcessing`:
- `seeAlso`: list of uri pointing to additional resources about the item
- `source*`: A sequence of characters giving the original source of the entity data as a URL. Recommended to be the fully qualified domain name of the source provider, or the URL to the source object.
- `typeOfAlgorithm*`: enumeration
- `version*`: Version of the model.

New properties by MARVEL

- `data*`: Payload containing extra ad-hoc complementary metadata needed by the MARVEL components not covered in the previous fields.

Required properties by SDM

- `id*`
- `type*`

Required properties by MARVEL

- `version*`

Data field contains metadata for the trained models:

- `file_name_weights*`: name of the object containing trained model parameters, the name is based on a convention agreed upon before (**https://git.marvel-project.eu/manolis.falelakis/marvel-technical-integration/-/issues/25**), `<component_name>-<out_format>-<partner>-<type>-<version_number>`
- `model_definition`: git url to the class file where the model is defined
- `code`: git url to the repository containing all the necessarry code to train the model
- `val_accuracy`: model performance (if model outputs compatible metric)
- `loss`: final traning loss (if model outputs it)
- `training_epochs`: number of epochs model was trained
- `...`: other relevant model metrics should also be added to the object in the data field, and other components can read specific information about the models from this field.

Examples

Example 1 (AVCC model, TensorFlow, HeatMap, DISCO dataset for training)

```
{
        "acceptableDataSources": [ "images" ], // images => spectrograms for audio
and frames from the video sources
        "algorithm": "avcc_tf",
        "alternateName": "AVCC model (heatmap output)", // redundant?
        "dataProvider": "UNS_F1",
        "dateCreated": "2022-05-29 08:25:29.988676",
        "dateModified": "2022-05-30 08:25:29.988676",
        "description": "Audio-Visual Crowd Counting model that outputs a heatmap
(density map)",
        "dockerImage": "registry.marvel-platform.eu/fedlclient_uns_f1:1", //
unclear, is this a docker image that built the model or the one that runs it,
because we can have one image that can run many models?
        "id": "avcc_heatmap_uns_f1_v0", // not sure if version should be included
in id, or some uuid or timestamp or similar?
        "inputAttributes": [ "audio_spectrogram", "video_frame" ],
        "mlFramework": "tensorflow==2.6.0", // version should be very important,
check?
        "name": "avcc_heatmap_uns_f1_v0", // redundant because of id?
        "outputAttributes": [ "heatmap" ], // just took heatmap from here:
https://git.marvel-project.eu/manolis.falelakis/marvel-technical-integration/-
/blob/main/Documentation/avcc.md
        "outputDataTypes": [ "image" ],
        "owner": [ "UNS1" ], // put all partners involved in the use case? unclear,
maybe should leave blank for now
        "refMLProcessing": ["https://git.marvel-
project.eu/manolis.falelakis/marvel-technical-integration/-
/blob/main/Documentation/avcc.md"], // based on https://github.com/smart-data-
models/dataModel.MachineLearning/blob/master/MLProcessing/README.md
        "seeAlso": ["https://git.marvel-project.eu/manolis.falelakis/marvel-
technical-integration/-/blob/main/Documentation/avcc.md"],
        "source": "https://marvel-platform.eu/files/shared/disco" // should be UNS
data, for now we can say this is the training data that was used to build this
model,
        "typeOfAlgorithm": "Deep Neural Network",
        "version": 0
        "data":  {
                "file_name_weights": "avcc-heatmap-uns-full_tensorflow-v0",
//<component_name>-<out_format>-<partner>-<type>-<version_number>
                "model_definition": "https://gitlab.au.dk/maleci/sl_vit/-
/blob/master/train_audio_visual_crowd_counting_backbone_stage1.ipynb", // uri to
gitlab, this is only the class which defines the model with all the imports
                "code": "https://gitlab.au.dk/maleci/sl_vit", // uri to gitlab
repo, all files that are needed to load/train the model
                "val_accuracy": 0.9621,
                "loss": 0.00123,
                "training_epochs": 100
                // can add any other information for training model
(hyperparameters, optimizer, etc.)
        }
        "type": "MLModel" // for SDM
}
```

Example 2 (AVCC model, TensorFlow, HeadCount, DISCO dataset for training)

```
{
        "acceptableDataSources": [ "images" ], // images => spectrograms for audio
and frames from the video sources
        "algorithm": "avcc_tf",
        "alternateName": "AVCC model (crowd counting output)", // redundant?
        "dataProvider": "UNS_F1",
        "dateCreated": "2022-05-29 08:25:29.988676",
        "dateModified": "2022-05-30 08:25:29.988676",
        "description": "Audio-Visual Crowd Counting model that outputs the people
count in the frame",
        "dockerImage": "registry.marvel-platform.eu/fedlclient_uns_f1:1", //
unclear, is this a docker image that built the model or the one that runs it,
because we can have one image that can run many models?
        "id": "avcc_headcount_uns_f1_v0", // not sure if version should be included
in id, or some uuid or timestamp or similar?
        "inputAttributes": [ "audio_spectrogram", "video_frame" ],
        "mlFramework": "tensorflow==2.6.0", // version should be very important,
check?
        "name": "avcc_headcount_uns_f1_v0", // redundant because of id?
        "outputAttributes": [ "predictedCount" ], // just took predictedCount from
here: https://git.marvel-project.eu/manolis.falelakis/marvel-technical-
integration/-/blob/main/Documentation/avcc.md
        "outputDataTypes": [ "integer" ],
        "owner": [ "UNS1" ], // put all partners involved in the use case? unclear,
maybe should leave blank for now
        "refMLProcessing": ["https://git.marvel-
project.eu/manolis.falelakis/marvel-technical-integration/-
/blob/main/Documentation/avcc.md"], // based on https://github.com/smart-data-
models/dataModel.MachineLearning/blob/master/MLProcessing/README.md
        "seeAlso": ["https://git.marvel-project.eu/manolis.falelakis/marvel-
technical-integration/-/blob/main/Documentation/avcc.md"],
        "source": "https://marvel-platform.eu/files/shared/disco" // should be UNS
data, for now we can say this is the training data that was used to build this
model,
        "typeOfAlgorithm": "Deep Neural Network",
        "version": 0
        "data":  {
                "file_name_weights": "avcc-headcount-uns-full_tensorflow-v0",
//<component_name>-<out_format>-<partner>-<type>-<version_number>
```

```
                "model_definition": "https://gitlab.au.dk/maleci/sl_vit/-
/blob/master/train_audio_visual_crowd_counting_backbone_stage2.ipynb", // uri to
gitlab, this is only the class which defines the model with all the imports
                "code": "https://gitlab.au.dk/maleci/sl_vit", // uri to gitlab
repo, all files that are needed to load/train the model
                "val_accuracy": 0.9621,
                "loss": 0.00123,
                "training_epochs": 100
                // can add any other information for training model
(hyperparameters, optimizer, etc.)
        }
        "type": "MLModel" // for SDM
}
```

Example 3 (AVCC model, PyTorch, HeatMap, DISCO dataset for training)

```
{
        "acceptableDataSources": [ "images" ], // images => spectrograms for audio
and frames from the video sources
        "algorithm": "avcc_pytorch",
        "alternateName": "AVCC model (heatmap output)", // redundant?
        "dataProvider": "UNS_F1",
        "dateCreated": "2022-05-29 08:25:29.988676",
        "dateModified": "2022-05-30 08:25:29.988676",
        "description": "Audio-Visual Crowd Counting model that outputs a heatmap
(density map)",
        "dockerImage": "registry.marvel-platform.eu/fedlclient_uns_f1:1", //
unclear, is this a docker image that built the model or the one that runs it,
because we can have one image that can run many models?
        "id": "avcc_heatmap_uns_f1_v0", // not sure if version should be included
in id, or some uuid or timestamp or similar?
        "inputAttributes": [ "audio_spectrogram", "video_frame" ],
        "mlFramework": "pytorch==1.8.2",
        "name": "avcc_heatmap_uns_f1_v0", // redundant because of id?
        "outputAttributes": [ "heatmap" ], // just took heatmap from here:
https://git.marvel-project.eu/manolis.falelakis/marvel-technical-integration/-
/blob/main/Documentation/avcc.md
        "outputDataTypes": [ "image" ],
        "owner": [ "UNS1" ], // put all partners involved in the use case? unclear,
maybe should leave blank for now
        "refMLProcessing": ["https://git.marvel-
project.eu/manolis.falelakis/marvel-technical-integration/-
/blob/main/Documentation/avcc.md"], // based on https://github.com/smart-data-
models/dataModel.MachineLearning/blob/master/MLProcessing/README.md
        "seeAlso": ["https://git.marvel-project.eu/manolis.falelakis/marvel-
technical-integration/-/blob/main/Documentation/avcc.md"],
        "source": "https://marvel-platform.eu/files/shared/disco" // should be UNS
data, for now we can say this is the training data that was used to build this
model,
        "typeOfAlgorithm": "Deep Neural Network",
        "version": 0
        "data":  {
                "file_name_weights": "avcc-heatmap-uns-full_pytorch-v0",
//<component_name>-<out_format>-<partner>-<type>-<version_number>
                "model_definition": "https://git.marvel-
project.eu/marvel/dynhp/dynhp-compressor/-/blob/master/models/l0_avcc.py", // uri
to gitlab, this is only the class which defines the model with all the imports
```

```
                "code: "https"://git.marvel-project.eu/marvel/dynhp/dynhp-
compressor", // uri to gitlab repo, all files that are needed to load/train the
model
                "val_accuracy": 0.9621,
                "loss": 0.00123,
                "training_epochs": 100
                // can add any other information for training model
(hyperparameters, optimizer, etc.)
        }
        "type": "MLModel" // for SDM
}
```

Example 4 (Repeated Example 1 with mandatory fields marked with "*")

```
{
        "acceptableDataSources*": [ "images" ], // images => spectrograms for audio
and frames from the video sources
        "algorithm*": "avcc_tf",
        "alternateName": "AVCC model (heatmap output)", // redundant?
        "dataProvider": "UNS_F1",
        "dateCreated*": "2022-05-29 08:25:29.988676",
        "dateModified*": "2022-05-30 08:25:29.988676",
        "description*": "Audio-Visual Crowd Counting model that outputs a heatmap
(density map)",
        "dockerImage": "registry.marvel-platform.eu/fedlclient_uns_f1:1", //
unclear, is this a docker image that built the model or the one that runs it,
because we can have one image that can run many models?
        "id*": "avcc_heatmap_uns_f1_v0", // not sure if version should be included
in id, or some uuid or timestamp or similar?
        "inputAttributes*": [ "audio_spectrogram", "video_frame" ],
        "mlFramework*": "tensorflow==2.6.0", // version should be very important,
check?
        "name*": "avcc_heatmap_uns_f1_v0", // redundant because of id?
        "outputAttributes*": [ "heatmap" ], // just took heatmap from here:
https://git.marvel-project.eu/manolis.falelakis/marvel-technical-integration/-
/blob/main/Documentation/avcc.md
        "outputDataTypes*": [ "image" ],
        "owner": [ "UNS1" ], // put all partners involved in the use case? unclear,
maybe should leave blank for now
        "refMLProcessing": ["https://git.marvel-
project.eu/manolis.falelakis/marvel-technical-integration/-
/blob/main/Documentation/avcc.md"], // based on https://github.com/smart-data-
models/dataModel.MachineLearning/blob/master/MLProcessing/README.md
        "seeAlso": ["https://git.marvel-project.eu/manolis.falelakis/marvel-
technical-integration/-/blob/main/Documentation/avcc.md"],
        "source*": "https://marvel-platform.eu/files/shared/disco" // should be UNS
data, for now we can say this is the training data that was used to build this
model,
        "typeOfAlgorithm*": "Deep Neural Network",
        "version*": 0
        "data*":  {
                "file_name_weights*": "avcc-heatmap-uns-full_tensorflow-v0",
//<component_name>-<out_format>-<partner>-<type>-<version_number>
                "model_definition": "https://gitlab.au.dk/maleci/sl_vit/-
/blob/master/train_audio_visual_crowd_counting_backbone_stage1.ipynb", // uri to
gitlab, this is only the class which defines the model with all the imports
                "code": "https://gitlab.au.dk/maleci/sl_vit", // uri to gitlab
repo, all files that are needed to load/train the model
```

```
                "val_accuracy": 0.9621,
                "loss": 0.00123,
                "training_epochs": 100
                // can add any other information for training model
(hyperparameters, optimizer, etc.)
        }
        "type*": "MLModel" // for SDM
}
```

CATFlow-Vehicles Raw Inference Result Data Model

Entity: CATFlow-Vehicles-Output

Global description:

A Data Model for the CATFlow-Vehicles inference model outputs of the MARVEL framework.

List of properties

List of properties defined as output by the inference model

- `location`: Location dictionary from settings file. It contains the following attributes:
    - `name`: The location name.
    - `id`: URL friendly location name using lowercase and dashes as per REST convention.
    - `uuid`: The UUID of the location.
    - `lat`: Latitude of camera location.
    - `lon`: Longitude of camera location.
- `camera`: Contains details about the camera. It contains the following attributes:
    - `type`: Hard-coded as "CAM". Consumer knows this part of the message is about the camera.
    - `group_id`: The camera group. Useful for virtual cameras.
    - `cam_id:` The name of the camera. Lowercase and dashes according to REST convention.
    - `cam_name`: The name of the camera. Lowercase and dashes according to REST convention.
    - `cam_uuid`: The UUID of the camera.
- `vehicle`: Contains details about the tracked vehicle. It contains the following attributes:
    - `type`: The label assigned to the tracked vehicle. ENUM: ["car", "bus", "light-goods-vehicle", "heavy-goods-vehicle", "bicycle", "motorcycle"]
    - `name`: A UI friendly name for the tracked vehicle (e.g: Light Goods Vehicle)
    - `entry`: Details about the entry of a vehicle. It contains the following attributes:
        - `ts`: Timestamp of entry
        - `carriageway_name`: Name of carriageway where vehicle entered
        - `carriageway_uuid`: UUID of the carriageway
        - `lane_name`: Name of the lane the vehicle entered
        - `lane_uuid`: The UUID of the lane
        - `lane_type`: Discerns what to track ("General" vs "Bicycle")
        - `lane_flow_uuid`: UUID of the lane flow, which connects exits and entries together
        - `uuid`: The UUID of the entry line
        - `handle`: The coordinates of the handle for the circle
    - `exit`: Details about the exit of a vehicle.
        - `ts`: Timestamp of entry
        - `carriageway_name`: Name of carriageway where vehicle entered
        - `carriageway_uuid`: UUID of the carriageway
        - `lane_name`: Name of the lane the vehicle entered
        - `lane_uuid`: The UUID of the lane
        - `lane_type`: Discerns what to track ("General" vs "Bicycle")

        - `lane_flow_uuid`: UUID of the lane flow, which connects exits and entries together
        - `uuid`: The UUID of the entry line
        - `handle`: The coordinates of the handle for the circle
    - `lane_flow_uuid`: UUID of the lane flow, which connects entries and exits together
    - `distance`: Format: Float - Range: >0 metres. Approximate distance between entry and exit
    - `time_seconds`: Format: Float - Range: 0 - 120 seconds. Time taken by vehicle to enter and exit
    - `speed_kmh`: Format: Float - Range: 0 - 200 km/h. Average speed of vehicle
    - `trajectory_points`: Format: Object with x and y image coordinates - Range: points between 0.0 and 1.0 since relative to image. List of points that visualise the path the vehicle took. Points represented as JSONs whose entries are relative to image size (e.g: {'x': 0.567, 'y': 0.356})
- `id`: Unique identifier of the entity. Formatted according to the NGSI standard. IDs for the especific media event can be automatically assigned by component.
- `MLModelId`: The id of the MLModel that generated this event. In this case it will be "CATFlow-V_v" and the version number of the model. It has to be one of the models defined in the registry.
- `name`: The name of this item. Typically in this case it could be the string "CATFlow-V-" plus a timestamp of the moment of the execution
- `owner`: A field containing a JSON encoded sequence of characters referencing the unique Ids of the owner(s). In the case of MARVEL, we can use the pilot name and use case here. ENUM: [GRN3, GRN4, MT1, MT3, UNS1].
- `detectedBy`: The id of the device that hosts the AI component instance. ENUM: [GRN_E1, GRN_E2, GRN_F1, GRN_F2, MT_F2]
- `type`: The type of the entity. In this case "MediaEvent"
- `startTime`: Timestamp when vehicle started getting tracked
- `endTime`: Timestamp when vehicle stopped getting tracked
- `dateProcessed`: The datetime when the object was detected and sent

Extra properties to be passed by the inference model to relate to the original data stream and other MARVEL elements

- `description`: A description of this item
- `id`: Unique identifier of the entity. Formatted according to the NGSI standard. IDs for the especific media event can be automatically assigned by component.
- `MLModelId`: The id of the MLModel that generated this event. In this case it will be "CATFlow-Vehicles_v" and the version number of the model. It has to be one of the models defined in the registry.
- `name`: The name of this item. Typically in this case it could be the string "CATFlow-Vehicle-" plus a timestamp of the moment of the execution
- `owner`: A field containing a JSON encoded sequence of characters referencing the unique Ids of the owner(s). In the case of MARVEL, we can use the pilot name and use case here. ENUM: [GRN3, GRN4, MT1, MT3, UNS1].
- `type`: The type of the entity. In this case "MediaEvent"
- `detectedBy`: Link to the device providing where the alert is detected (the link to the device id of the infrastructure in MARVEL)

Required properties by MARVEL

- `vehicle:type`
- `time_seconds`
- `distance`
- `speed_kmh`
- `trajectory_points`
- `id`
- `owner`
- `camera:cam_id` NOTE: or another property that matches with the cameraId in MARVEL
- `dateProcessed`
- `MLModelId`
- `entry:TS` NOTE: Equivalent to the startTime in other models
- `exitTS` NOTE: Equivalent to the endTime in other models
- `detectedBy`

Example payload

```
{
    "location": {
        "name": "Mġarr - iż-Żebbiegh",
        "id": "mgarr-iz-zebbiegh",
        "uuid": "f630a888-6f40-4784-b377-e1c75f2b849b",
        "lat": "35.91877777777778",
        "lon": "14.376527777777778"
    },
    "camera": {
        "type": "CAM",
        "group_id": "[cam-grn-va-01]",
        "cam_id": "Cam-GRN-VA-01",
        "cam_name": "Cam-GRN-VA-01",
        "cam_uuid": "45a767a7-9bfc-4b13-a459-6432d30c6028"
    },
    "vehicle": {
        "type": "car",
        "name": "Car",
        "entry": {
            "ts": "2022-02-20T10:46:43.331Z",
            "carriageway_name": "Mġarr - Triq iż-Żebbiegh",
            "carriageway_uuid": "a31d0d53-be33-49db-8b24-ba81e3c84509",
            "lane_name": "Mġarr - Triq iż-Żebbiegh - Eastwards",
            "lane_uuid": "a6278135-95db-44f0-a91a-0694ad551999",
            "lane_type": "general",
            "uuid": "3a8fb5a2-f47c-45d4-872d-7acdf17cd96d"
        },
        "exit": {
            "ts": "2022-02-20T10:46:45.205Z",
            "carriageway_name": "Mġarr - Triq iż-Żebbiegh",
            "carriageway_uuid": "a31d0d53-be33-49db-8b24-ba81e3c84509",
            "lane_name": "Mġarr - Triq iż-Żebbiegh - Eastwards",
            "lane_uuid": "a6278135-95db-44f0-a91a-0694ad551999",
            "lane_type": "general",
            "uuid": "f26e2a40-f9c0-4d46-accf-501f4444fdd6"
        },
        "lane_flow_uuid": "65219a0f-c924-48ca-a007-e1c9a057dd14",
        "distance": 24,
        "time_seconds": 1.874,
        "speed_kmh": 46.10458911419423,
        "trajectory_points": [
            {
                "x": 0.6627604166666666,
                "y": 0.3550925925925925 7
            },
            {
                "x": 0.5627604166666667,
                "y": 0.4319444444444446
            },
            {
                "x": 0.4619791666666666,
                "y": 0.5143518518518518
            },
            {
                "x": 0.3625,
                "y": 0.59375
            },
            {
                "x": 0.2578125,
                "y": 0.6731481481481482
            },
            {
                "x": 0.1875,
                "y": 0.7261574074074074
            },
            {
                "x": 0.08645833333333333,
                "y": 0.8
            }
        ]
    },
    "id": "b8128b96-242e-441a-925b-3a9c2dd26eb5",
    "MLModelId": "CATFlow-V_v01",
    "name": "CATFlow-V2023-04-11T07:58:19.474Z",
    "owner": "GRN",
    "detectedBy": "GRN_E1",
    "type": "MediaEvent",
    "dateProcessed": "2023-04-11T07:58:19.474Z",
    "startTime": "2022-02-20T10:46:43.331Z",
    "endTime": "2022-02-20T10:46:45.205Z",
    "cameraId": "Cam-GRN-VA-01"
}
```

CATFlow-Pedestrians Raw Inference Result Data Model

Entity: CATFlow-Pedestrians-Output

Global description:

A Data Model for the CATFlow-Pedestrians inference model outputs of the MARVEL framework.

List of properties

List of properties defined as output by the inference model

- `location`: Location dictionary from settings file. It contains the following attributes:
    - `name`: The location name.
    - `id`: URL friendly location name using lowercase and dashes as per REST convention.
    - `uuid`: The UUID of the location.
    - `lat`: Latitude of camera location.
    - `lon`: Longitude of camera location.
- `camera`: Contains details about the camera. It contains the following attributes:
    - `type`: Hard-coded as “CAM”. Consumer knows this part of the message is about the camera.
    - `group_id`: The camera group. Useful for virtual cameras.
    - `cam_id:` The name of the camera. Lowercase and dashes according to REST convention.
    - `cam_name`: The name of the camera. Lowercase and dashes according to REST convention.
    - `cam_uuid`: The UUID of the camera.
- `pedestrian`: Contains details about the tracked pedestrian. It contains the following attributes:
    - `start_ts`: Timestamp when pedestrian started getting tracked
    - `end_ts`: Timestamp when pedestrian stopped getting tracked
    - `time_seconds`: Format: Float - Range: 0 - 120 seconds. Time taken by pedestrian to enter and exit of the scene
    - `trajectory_points`: Format: Object with x and y image coordinates - Range: points between 0.0 and 1.0 since relative to image. List of points that visualise the path the pedestrian took. Points represented as JSONs whose entries are relative to image size (e.g: {‘x’: 0.567, ‘y’: 0.356})
- `id`: Unique identifier of the entity. Formatted according to the NGSI standard. IDs for the especific media event can be automatically assigned by component.
- `MLModelId`: The id of the MLModel that generated this event. In this case it will be "CATFlow-P_v" and the version number of the model. It has to be one of the models defined in the registry.
- `name`: The name of this item. Typically in this case it could be the string "CATFlow-P" plus a timestamp of the moment of the execution
- `owner`: A field containing a JSON encoded sequence of characters referencing the unique Ids of the owner(s). In the case of MARVEL, we can use the pilot name and use case here. ENUM: [GRN3, GRN4, MT1, MT3, UNS1].
- `detectedBy`: The id of the device that hosts the AI component instance. ENUM: [GRN_E1, GRN_E2, GRN_F1, GRN_F2, MT_F2]

- type: The type of the entity. In this case "MediaEvent"
- startTime: Timestamp when pedestrian started getting tracked
- endTime: Timestamp when pedestrian stopped getting tracked
- dateProcessed: The datetime when the object was detected and sent

Extra properties to be passed by the inference model to relate to the original data stream and other MARVEL elements

- description: A description of this item
- id: Unique identifier of the entity. Formatted according to the NGSI standard. IDs for the especific media event can be automatically assigned by component.
- MLModelId: The id of the MLModel that generated this event. In this case it will be "CATFlow-P_v" and the version number of the model. It has to be one of the models defined in the registry.
- name: The name of this item. Typically in this case it could be the string "CATFlow-P" plus a timestamp of the moment of the execution
- owner: A field containing a JSON encoded sequence of characters referencing the unique Ids of the owner(s). In the case of MARVEL, we can use the pilot name and use case here. ENUM: [GRN3, GRN4, MT1, MT3, UNS1].
- type: The type of the entity. In this case "MediaEvent"
- detectedBy: Link to the device providing where the alert is detected (the link to the device id of the infrastructure in MARVEL)

Required properties by MARVEL

- time_seconds
- itrajectory_points
- id
- owner
- camera:cam_id NOTE: or another property that matches with the cameraId in MARVEL
- dateProcessed
- MLModelId
- startTS
- endTS
- detectedBy

Example payload

```
{
    "location": {
        "name": "Mġarr - iż-Żebbiegh",
        "id": "mgarr-iz-zebbiegh",
        "uuid": "f630a888-6f40-4784-b377-e1c75f2b849b",
        "lat": "35.91877777777778",
        "lon": "14.376527777777778"
    },
    "camera": {
        "type": "CAM",
        "group_id": "[cam-grn-va-01]",
        "cam_id": "Cam-GRN-VA-01",
        "cam_name": "Cam-GRN-VA-01",
        "cam_uuid": "45a767a7-9bfc-4b13-a459-6432d30c6028"
    },
    "pedestrian": {
        "start_ts": "2022-02-20T10:42:21.683Z",
        "end_ts": "2022-02-20T10:42:26.547Z",
        "time_seconds": 4.864,
        "trajectory_points": [
            {
                "x": 0.08463541666666667,
                "y": 0.6601851851851852
            },
            {
                "x": 0.13385416666666666,
                "y": 0.6305555555555555
            }
        ]
    },
    "id": "15adec18-2335-49b1-ad42-a59ea8e0e84c",
    "MLModelId": "CATFlow-P_v01",
    "name": "CATFlow-P2023-04-11T07:38:56.536Z",
    "owner": "GRN",
    "detectedBy": "GRN_E1",
    "type": "MediaEvent",
    "dateProcessed": "2023-04-11T07:38:56.536Z",
    "startTime": "2022-02-20T10:42:21.683Z",
    "endTime": "2022-02-20T10:42:26.547Z",
    "cameraId": "Cam-GRN-VA-01"
}
```

TAD Raw Inference Result Data Model

Entity: TAD-Output

Global description:

A Data Model for the TAD inference model outputs of the MARVEL framework.

List of properties

List of properties defined as output by the inference model

- `entryTs`: data type: float (seconds), entry timestamp pointing to the absolute time in the video. Following the ISO8601 UTC format.
- `exitTs`: data type: float (seconds), exit timestamp pointing to the absolute time in the video. Following the ISO8601 UTC format.
- `speed`: data type: string, value of the anomalous speed
- `category`: data type: string, a field containing the catefory of the anomaly detected. In the case of TAD, it would be low or high speed. ENUM: [HighSpeed, LowSpeed].
- `cameraId`: Unique identifier of the camera from where the video was taken. Formatted according to the NGSI standard (i.e. URN identifier). IDs for AV sources will be hard-coded manually in a configuration file or automatically assigned by the AV registry component and will include a 12-digit hexadecimal identifier.
- `dateProcessed`: Timestamp of the processing of the entity by the inference model.
- `description`: A description of this item
- `id`: Unique identifier of the entity. Formatted according to the NGSI standard. IDs for the especific media event can be automatically assigned by component.
- `MLModelId`: The id of the MLModel that generated this event. In this case it will be "TAD-" and the version number of the model. It has to be one of the models defined in the registry.
- `name`: The name of this item. Typically in this case it could be the string "TAD-" plus a timestamp of the moment of the execution
- `owner`: A field containing a JSON encoded sequence of characters referencing the unique Ids of the owner(s). In the case of MARVEL, we can use the pilot name and use case here. ENUM: [GRN3, GRN4, MT1, MT3, UNS1].
- `thresholdBreach` : Description of an observed threshold breach that contributed to detection of an anomaly
- `type`: The type of the entity. In this case "Anomaly"

Extra properties to be passed by the inference model to relate to the original data stream and other MARVEL elements

- `cameraId`: Unique identifier of the camera from where the video was taken. Formatted according to the NGSI standard (i.e. URN identifier). IDs for AV sources will be hard-coded manually in a configuration file or automatically assigned by the AV registry component and will include a 12-digit hexadecimal identifier.
- `dateProcessed`: Timestamp of the processing of the entity by the inference model.
- `description`: A description of this item
- `id`: Unique identifier of the entity. Formatted according to the NGSI standard. IDs for the especific media event can be automatically assigned by component.
- `catflowid`: Unique identifier of the entity from the CATFlow output, such that entries can be tracked to the CATFlow output. Formatted according to the NGSI standard. IDs for the especific media event can be automatically assigned by component.

- `trajectory_points`: Format: Object with x and y image coordinates - Range: points between 0.0 and 1.0 since relative to image. List of points that visualise the path the vehicle took. Points represented as JSONs whose entries are relative to image size (e.g: {'x': 0.567, 'y': 0.356})
- `MLModelId`: The id of the MLModel that generated this event. In this case it will be "TAD-" and the version number of the model. It has to be one of the models defined in the registry.
- `name`: The name of this item. Typically in this case it could be the string "TAD-" plus a timestamp of the moment of the execution
- `owner`: A field containing a JSON encoded sequence of characters referencing the unique Ids of the owner(s). In the case of MARVEL, we can use the pilot name and use case here. ENUM: [GRN3, GRN4, MT1, MT3, UNS1].
- `thresholdBreach` : Description of an observed threshold breach that contributed to detection of an anomaly
- `type`: The type of the entity. In this case "Anomaly"
- `detectedBy`: Link to the device providing where the alert is detected (the link to the device id of the infrastructure in MARVEL)

Required properties by MARVEL

- `startTime`
- `endTime`
- `speed`
- `category`
- `id`
- `type`
- `owner`
- `cameraId`
- `dateProcessed`
- `MLModelId`
- `detectedBy`

Example payload

```
{
        "entry_Ts": "17-05-2022T04:59:14",
        "exit_Ts": "2022-05-17T06:59:11.067605+02:00",
        "speed": 96.75598677668181,
        "category": "HighSpeed",
        "cameraId": "greenroads-streams1-stream1",
        "dateProcessed": "17-05-2022T04:59:14",
        "description": "Speed anomaly detector from CATFlow output",
        "id": "TAD_Event-17-05-2022T04:59:14",
        "catflowid": "41dd90c8-b029-47bf-acff-1c5ec6ee6736",
        "trajectory_points": [
                {
                    "x": 0.16458333333333333,
                    "y": 0.687037037037037
                },
                {
                    "x": 0.19583333333333333,
                    "y": 0.6722222222222223
                },
                {
                    "x": 0.22604166666666667,
                    "y": 0.6472222222222223
                },
                {
                    "x": 0.25729166666666664,
                    "y": 0.6268518518518519
                },
                {
                    "x": 0.28854166666666664,
                    "y": 0.6101851851851852
                },
                {
                    "x": 0.321875,
                    "y": 0.5935185185185186
                },
                {
                    "x": 0.3541666666666667,
                    "y": 0.5805555555555556
                },
                {
                    "x": 0.3854166666666667,
                    "y": 0.5712962962962963
                },
                {
                    "x": 0.42109375,
                    "y": 0.5613425925925926
                },
                {
                    "x": 0.45416666666666666,
                    "y": 0.5652777777777778
                },
                {
                    "x": 0.4864583333333333,
                    "y": 0.5671296296296297
                },
                {
                    "x": 0.5166666666666667,
                    "y": 0.5745370370370371
                },
                {
                    "x": 0.55,
                    "y": 0.5856481481481481
                },
                {
                    "x": 0.5854166666666667,
                    "y": 0.600462962962963
                },
                {
                    "x": 0.5916666666666667,
                    "y": 0.6023148148148149
                }
            ]

        "MLModelId": "TAD-v01",
        "name": "TAD-17-05-2022T04:59:14",
        "owner": "GRN",
        "thresholdBreach": 81.33301819276505,
        "type": "Anomaly"
}
```

ViAD / AVAD Raw Inference Result Data Model

Entity: ViAD/AVAD-Output

Global description:

A Data Model for the ViAD and AVAD inference model outputs of the MARVEL framework.

List of properties

List of properties defined as output by the inference model

- `predictedAnomaly`: data type: boolean, true or false, depending if the anomaly is detected (NOTE - Is this necessary? If the anomaly is not detected, it would suffice not to send any output).
- `startTime`: data type: float (seconds), absolute timestamp for the event start. Following the ISO8601 UTC format.
- `endTime`: data type: float (seconds), absolute timestamp for the event end. Following the ISO8601 UTC format.

Extra properties to be passed by the inference model to relate to the original data stream and other MARVEL elements

- `cameraId`: Unique identifier of the camera from where the video was taken. Formatted according to the NGSI standard (i.e. URN identifier). IDs for AV sources will be hard-coded manually in a configuration file or automatically assigned by the AV registry component and will include a 12-digit hexadecimal identifier.
- `dateProcessed`: Timestamp of the processing of the entity by the inference model.
- `description`: A description of this item
- `id`: Unique identifier of the entity. Formatted according to the NGSI standard. IDs for the especific media event can be automatically assigned by component.
- `MLModelId`: The id of the MLModel that generated this event. In this case it will be "AVAD-" and the version number of the model. It has to be one of the models defined in the registry.
- `name`: The name of this item. Typically in this case it could be the string "AVAD-" plus a timestamp of the moment of the execution
- `owner`: A field containing a JSON encoded sequence of characters referencing the unique Ids of the owner(s). In the case of MARVEL, we can use the pilot name and use case here. ENUM: [GRN3, GRN4, MT1, MT3, UNS1].
- `type`: The type of the entity. In this case "Anomaly"
- `detectedBy`: Link to the device providing where the alert is detected (the link to the device id of the infrastructure in MARVEL)

Required properties by MARVEL

- `predictedAnomaly`
- `id`
- `type`
- `owner`
- `cameraId`
- `dateProcessed`
- `MLModelId`
- `startTime`
- `endTime`
- `detectedBy`

Example payload

```
{
  "predictedAnomaly": "true",
  "id": "AVAD_Event_2022-04-11T06:40:10.063Z",
  "type": "Anomaly",
  "cameraId": "Cam-GRN-VA-01",
  "dateProcessed": "2022-04-11T06:42:10.063Z",
  "name": "AVAD-2022-04-11T06:40:10.063Z",
  "MLModelId": "AVAD-v1",
  "startTime": "2022-04-11T06:42:10.063Z",
  "endTime": "2022-04-11T06:42:10.063Z",
  "detectedBy": "GRN_Fog_Server"
}
```

VCC / AVCC Raw Inference Result Data Model

Entity: VCC/AVCC-Output

Global description:

A Data Model for the VCC and AVCC inference model outputs of the MARVEL framework.

List of properties

List of properties defined as output by the inference model

- `predictedCount`: data type: string, number of people counted by the AI model in the frame
- `heatmap`: data type: string, image encoded using Base64.

Extra properties to be passed by the inference model to relate to the original data stream and other MARVEL elements

- `cameraId`: Unique identifier of the camera from where the video was taken. Formatted according to the NGSI standard (i.e. URN identifier). IDs for AV sources will be hard-coded manually in a configuration file or automatically assigned by the AV registry component and will include a 12-digit hexadecimal identifier.
- `dateProcessed`: Timestamp of the processing of the entity by the inference model.
- `description`: A description of this item
- `id`: Unique identifier of the entity. Formatted according to the NGSI standard. IDs for the especific media event can be automatically assigned by component.
- `MLModelId`: The id of the MLModel that generated this event. In this case it will be "AVCC-" and the version number of the model. It has to be one of the models defined in the registry.
- `name`: The name of this item. Typically in this case it could be the string "AVCC-" plus a timestamp of the moment of the execution
- `owner`: A field containing a JSON encoded sequence of characters referencing the unique Ids of the owner(s). In the case of MARVEL, we can use the pilot name and use case here. ENUM: [GRN3, GRN4, MT1, MT3, UNS1].
- `type`: The type of the entity. In this case "MediaEvent"
- `timestamp`: Entity creation timestamp. In MARVEL, this is the absolute timestamp pointing to the exact time of the video stream to which this element is referring to. Optional. Fill-in if the event is referring to specific time (either this field or starTime/endTime should be popultated).
- `detectedBy`: Link to the device providing where the alert is detected (the link to the device id of the infrastructure in MARVEL)
- `secondCameraId`: The id of a second Camera entity (audio or video) that produced a AV stream. This is used only for some inference components that need to digest two streams (audio / video) at the same time, and it is complementary and following the same conventions as the cameraId (main stream). Used so far for VCC and AVCC, not mandatory.

Required properties by MARVEL

- `predictedCount`
- `id`
- `type`
- `owner`
- `cameraId`
- `timestamp`
- `dateProcessed`
- `MLModelId`
- `detectedBy`

Example payload

```
{
  "predictedCount": "14.0",
  "id": "AVCC_Event_2022-04-11T06:40:10.063Z",
  "type": "MediaEvent",
  "cameraId": "Cam-MGARR",
  "timestamp": "2022-04-11T06:35:20.065Z",
  "dateProcessed": "2022-04-11T06:40:10.063Z",
  "name": "AVCC-2022-04-11T06:40:10.063Z",
  "MLModelId": "AVCC-v1",
  "detectedBy": "GRN_Fog_Server",
  "heatmap":
"b'/9j/4AAQSkZJRgABAQAAAQABAAD/2wBDAAIBAQEBAQIBAQECAgICAgQDAgICAgUEBAMEBgUGBgYFBgYG
BwkIBgcJBwYGCAsICQoKCgoKBggLDAsKDAkKCgr/2wBDAQICAgICAgUDAwUKBwYHCgoKCgoKCgoKCgoKCgo
KCgoKCgoKCgoKCgoKCgoKCgoKCgoKCgoKCgoKCgr/wAARCAJABAADASIAAhEBAxEB/8QAHwAAAQ
UBAQEBAQEAAAAAAAAAAAECAwQFBgcICQoL/8QAtRAAAgEDAwIEAwUFBAQAAAF9AQIDAAQRBRIhMUEGE1FhB
yJxFDKBkaEII0KxwRVS0fAkM2JyggkKFhcYGRolJicoKSo0NTY3ODk6Q0RFRkdISUpTVFVWV1hZWmNkZWZn
aGlqc3R1dnd4eXqDhIWGh4iJipKTlJWWl5iZmqKjpKWmp6ipqrKztLW2t7i5usLDxMXGx8jJytLT1NXW19j
Z2uHi4+Tl5ufo6erx8vP09fb3+Pn6/8QAHwEAAwEBAQEBAQEBAQAAAAAAAAECAwQFBgcICQoL/8QAtREAAg
ECBAQDBAcFBAQAAQJ3AAECAxEEBSExBhJBUQdhcRMiMoEIFEKRobHBCSMzUvAVYnLRChYkNOEl8RcYGRomJ
ygpKjU2Nzg5OkNERUZHSElKU1RVVldYWVpjZGVmZ2hpanN0dXZ3eHl6goOEhYaHiImKkpOUlZaXmJmaoqOk
paanqKmqsrO0tba3uLm6wsPExcbHyMnK0tPU1dbX2Nna4uPk5ebn6Onq8vP09fb3+Pn6/9oADAMBAAIRAxE
APwD8KdT1DWtd1KTVdW3T3EpBkkePG7AA7AdgK0/DHiLW9AdzpthahpMAmXT45SOvTzFbHXtX0Y5BclRxni
kr9cpcc4zC1G6cNerunf74n5RLi+lOl7OWFXL25tPu5bGB8J/2uvjb8OVlk0EqDNtOY9NtV27d3/TA5+8fp
XfwftSeDdamFzrP7NU91KB8rnxxfLgHqflGDwMdK5+ivFzjOMqz3E/WMZgKcqlrOVoqTXROUYJtLom+/dme
B4seV831GFShzO8lRxFWkpPvJU3FNro2nbpuzvLH9p7wa0ieb+zZPuQgqx8cXvH/AI57elei6N+3b8Vtbsb
TRdO0UaZaW4YSRC5SfzMsrD5pIdwwQe/f2r5+orzsJUyDAYuGKw+X0Kdam7xnGlTjKL/mlSik/1mV
Y0fY4729eH8tXFV6sL7puE5Si2nZq60aTWqR71omrteQR+a44z1Yf57VvWb24IDTx/XIr5nor3q/FtfE1H
q07ye75LLf/nVM05ffNJ5LXGttFh//ACb/AO1Pqq1kt1PAQk/BTjE8r+iqL1ysO4rGuwQlPAO5Sd0nTDGD/
Ib0tKtpuJDcgf9ZPBb0KukF3Qu+1HbB9ZPj+RO8EZxHfw2kZpJkRU5cFtlOgPfmG+PO11cWvQifcn8rxGU9v
AeU7c+KPiN3zq1rdP1PA1tKH4ElMtKfGN86RJvz2dL8ItMBU3PF4E3LFNEOx09NqLcYjgTQXMxOuYc4jgRtfe
q6lvkZf0LfUUlDcp+TWUPL1n+zOvE50MSSgVSLDULhAlWLcaFKSNl8nUDoVBOoBJ5V/aDHV+x0gjrzbbA3X2
KxFBQwYuCxsl3MLtY1TvA+DzZB+u3AdcPZ3vHyRXhWUWzUkyLZ4+S5VCDGSAmIh8WBVmyOhgwvrT7ZXzq
LaCNlrcFh23Dqs6Ud7u5ZiLuv6+1w0TLMvfyjqtGGtuWgGeUnmSAnD49drELiblxMLfF5VhL/jOwK1P2Z6o
jv3s7JSKFpvxYGRNE8vXJTD1ZE56V5ocU+QuPvBbk8e5i0Q4PdBadSO2YLWQXqYvXAiX+UUAHLqbCmn7r9h
5vkycoV/lgTi1Bz/t7B+yQRqBCx/pDlkgoKw7pMNq+xNrnJ+XzTJSFFpqw8Hbi+/o9OZsb0rUSMaJlJqNUqDN
OiEQqtmJbKpZ0ah7yi3Wd6jhJtbjQ8GzKzmWydjv2DK2zP9UMPVi4Jg7bpazx6Qr1JPkeB9G8Pmn6R3xsjMl
Z3CL+f5mpZn/pnXxfLjY6MsJWTfE8f6lrrTyxGr0ZOvfJ0vPKjZjNjmFShhM9G4Mw4D+a0H3xkiF8vmp0Hh
qIsSjG+CC2wEjUwaFuk0dqKAQqFWOrxzzrn4WZwbPeK0X/v3cbFqVUgA/FgqtiOv0sHsOZNzkxAXGHD+cvJ
8ueWv6sjgkPSQjFT9KyV5ZFUmHCSH4SrN1Sr0a1Ly8wXNUP5hU6zo8Oz3fvIqxYpeG1M4Ek4sR9q2t87PmDR
nNeNdp4WGjfwdcHO8SC6kP0Z/7bVb0GKX3hNRi02wVPRjAh3kNOEwtLnNlpDgRt0vm19ZqpcdGVWKodPEL3
YSXjIXtdaBhuvjVLg7hBb0Qh6Ej+iJ1JqYJDGbNXHJx+9mB9htMt1CLTVRlAdfmaE5W7KyGtNYhpq6lHU1fXr
X8d7AC1uSsTQnPRVqLp4lqSK5XmOnnzIkBmXXBfMBLVLDttm7yKC+nbjKRubIrc3ZjYcV3c3aRJLHEr5B4V
JIj2tREM7JL80L7FV6CHKwcrxq9RIPLN6XcXqGtVqKL+Z4GFNFaXw5JYnhxRp0Q3Atk8IJJrSGJoW7e65YNkI
ZJKIuU7h3oC1HEphKXyGH3K1e9oUoHMKPPvhlI9MmN+sJjGrY+Y1xULWpaCc9t9XNbL0X2A4kp41Rnz+wJ5
H0VeHaW/bD6AD0ZKq3wnnhz/dNFKQX+IIjQ9rVJcsr7C/G+ioAIBw0k5QzGHiIgaZy2vCzJHqNeiIqkLyfnDF
```

SED Raw Inference Result Data Model

Entity: SED-Output

Global description:

A Data Model for the SED inference model outputs of the MARVEL framework.

List of properties

List of properties defined as output by the inference model

- `startTime`: data type: float (seconds), absolute timestamp for the event start. Following the ISO8601 UTC format.
- `endTime`: data type: float (seconds), absolute timestamp for the event end. Following the ISO8601 UTC format.
- `label`: data type: string, value range defined by use case, for GRN4 ["motorcycle", "bicycle", "car", "bus"].

Extra properties to be passed by the inference model to relate to the original data stream and other MARVEL elements

- `cameraId`: Unique identifier of the camera from where the video was taken. Formatted according to the NGSI-LD standard (i.e. URN identifier). IDs for AV sources will be hard-coded manually in a configuration file or automatically assigned by the AV registry component and will include a 12-digit hexadecimal identifier.
- `dateProcessed`: Timestamp of the processing of the entity by the inference model.
- `description`: A description of this item
- `id`: Unique identifier of the entity. Formatted according to the NGSI standard. IDs for the especific media event can be automatically assigned by component.
- `MLModelId`: The id of the MLModel that generated this event. In this case it will be "SED-" and the version number of the model. It has to be one of the models defined in the registry.
- `name`: The name of this item. Typically in this case it could be the string "SED_" plus a timestamp of the moment of the execution
- `owner`: A field containing a JSON encoded sequence of characters referencing the unique Ids of the owner(s). In the case of MARVEL, we can use the pilot name and use case here. ENUM: [GRN3, GRN4, MT1, MT3, UNS1].
- `type`: The type of the entity. In this case "MediaEvent"
- `detectedBy`: Link to the device providing where the alert is detected (the link to the device id of the infrastructure in MARVEL)

Required properties by MARVEL

- `startTime`
- `endTime`
- `label`
- `id`
- `type`
- `owner`
- `cameraId`
- `dateProcessed`
- `MLModelId`
- `detectedBy`

Example payload

```
{
  "owner": "GRN4",
  "startTime": "2022-04-11T06:40:10.063Z",
  "endTime": "2022-04-11T06:40:10.063Z",
  "label": "bus",
  "id": "SEDEvent_2022-04-11T06:40:10.063Z",
  "type": "MediaEvent",
  "cameraId": "Cam-MGARR",
  "dateProcessed": "2022-04-11T06:40:10.063Z",
  "name": "SED-2022-04-11T06:40:10.063Z",
  "MLModelId": "SED-v1",
  "detectedBy": "GRN_Fog_Server"
}
```

AT Raw Inference Result Data Model

Entity: AudioTagging-Output

Global description:

A Data Model for the AudioTagging inference model outputs of the MARVEL framework. In case of Audio Tagging multiple classes can be active at given time, and onset-offset time are not overlapping with other events. Event onsets and offset are on fixed grid. For audio tagging, the output is again similar having multiple events active at the same time instance.

List of properties

List of properties defined as output by the inference model

- `startTime`: data type: float (seconds), absolute timestamp for the event start. Following the ISO8601 UTC format.
- `endTime`: data type: float (seconds), absolute timestamp for the event end. Following the ISO8601 UTC format.
- `label`: data type: string, value range defined by use case, ENUM to be defined, including for instance "TrafficLight", "LowSpeed", "HighSpeed", "ModerateSpeed", "SparseTraffic", etc.

Extra properties to be passed by the inference model to relate to the original data stream and other MARVEL elements

- `cameraId`: Unique identifier of the camera from where the video was taken. Formatted according to the NGSI-LD standard (i.e. URN identifier). IDs for AV sources will be hard-coded manually in a configuration file or automatically assigned by the AV registry component and will include a 12-digit hexadecimal identifier.
- `dateProcessed`: Timestamp of the processing of the entity by the inference model.
- `description`: A description of this item
- `id`: Unique identifier of the entity. Formatted according to the NGSI standard. IDs for the especific media event can be automatically assigned by component.
- `MLModelId`: The id of the MLModel that generated this event. In this case it will be "SED-" and the version number of the model. It has to be one of the models defined in the registry.
- `name`: The name of this item. Typically in this case it could be the string "SED_" plus a timestamp of the moment of the execution
- `owner`: A field containing a JSON encoded sequence of characters referencing the unique Ids of the owner(s). In the case of MARVEL, we can use the pilot name and use case here. ENUM: [GRN3, GRN4, MT1, MT3, UNS1].
- `timestamp`: Reported timestamp pointing to the absolute time in the video where the observation was made
- `type`: The type of the entity. In this case "MediaEvent"
- `detectedBy`: Link to the device providing where the alert is detected (the link to the device id of the infrastructure in MARVEL)

Required properties by MARVEL

- `startTime`
- `endTime`
- `label`
- `id`
- `type`
- `owner`
- `cameraId`
- `detectedBy`

- dateProcessed
- MLModelId

Example payload

```
{
  "owner": "GRN4",
  "startTime": "2022-04-11T06:35:20.065Z",
  "endTime":"2022-04-11T06:35:24.065Z",
  "label": "TrafficLights",
  "id": "ATEvent_2022-04-11T06:45:20.063Z",
  "type": "MediaEvent",
  "cameraId": "Cam-MGARR",
  "dateProcessed": "2022-04-11T06:45:20.063Z",
  "name": "AT-2022-04-11T06:40:10.063Z",
  "detectedBy": "GRN_Fog_Server",
  "MLModelId": "AT-v1"
}
{
  "owner": "GRN4",
  "startTime": "2022-04-11T06:35:20.065Z",
  "endTime":"2022-04-11T06:35:24.065Z",
  "label": "LowSpeed",
  "id": "ATEvent_2022-04-11T06:45:22.068Z",
  "type": "MediaEvent",
  "cameraId": "Cam-MGARR",
  "dateProcessed": "2022-04-11T06:45:22.068Z",
  "name": "AT-2022-04-11T06:40:10.063Z",
  "detectedBy": "GRN_Fog_Server",
  "MLModelId": "AT-v1"
}
{
  "owner": "GRN4",
  "startTime": "2022-04-11T06:35:24.065Z",
  "endTime":"2022-04-11T06:35:28.065Z",
  "label": "SparseTraffic",
  "id": "ATEvent_2022-04-11T06:45:23.057Z",
  "type": "MediaEvent",
  "cameraId": "Cam-MGARR",
  "dateProcessed": "2022-04-11T06:45:23.057Z",
  "name": "AT-2022-04-11T06:40:10.063Z",
  "detectedBy": "GRN_Fog_Server",
  "MLModelId": "AT-v1"
}
```

AAC Raw Inference Result Data Model

Entity: AAC-Output

Global description:

A Data Model for the AAC inference model outputs of the MARVEL framework.

List of properties

List of properties defined as output by the inference model

- `startTime`: data type: float (seconds), absolute timestamp for the event start. Following the ISO8601 UTC format.
- `endTime`: data type: float (seconds), absolute timestamp for the event end. Following the ISO8601 UTC format.
- `label`: data type: string, text description for the given segment, multiple words, free form.

Extra properties to be passed by the inference model to relate to the original data stream and other MARVEL elements

- `cameraId`: Unique identifier of the camera from where the video was taken. Formatted according to the NGSI-LD standard (i.e. URN identifier). IDs for AV sources will be hard-coded manually in a configuration file or automatically assigned by the AV registry component and will include a 12-digit hexadecimal identifier.
- `dateProcessed`: Timestamp of the processing of the entity by the inference model.
- `description`: A description of this item
- `id`: Unique identifier of the entity. Formatted according to the NGSI standard. IDs for the especific media event can be automatically assigned by component.
- `MLModelId`: The id of the MLModel that generated this event. In this case it will be "SED-" and the version number of the model. It has to be one of the models defined in the registry.
- `name`: The name of this item. Typically in this case it could be the string "SED_" plus a timestamp of the moment of the execution
- `owner`: A field containing a JSON encoded sequence of characters referencing the unique Ids of the owner(s). In the case of MARVEL, we can use the pilot name and use case here. ENUM: [GRN3, GRN4, MT1, MT3, UNS1].
- `type`: The type of the entity. In this case "MediaEvent"
- `detectedBy`: Link to the device providing where the alert is detected (the link to the device id of the infrastructure in MARVEL)

Required properties by MARVEL

- `startTime`
- `endTime`
- `label`
- `id`
- `type`
- `owner`
- `cameraId`
- `dateProcessed`
- `MLModelId`
- `detectedBy`

Example payload

```
{
  "owner": "GRN4",
  "startTime": 2022-04-11T06:40:10.063Z,
  "endTime":2022-04-11T06:40:14.063Z,
  "caption": "A bus passes by while it is raining.",
  "id": "AACEvent_2022-04-11T06:40:10.063Z",
  "type": "MediaEvent",
  "cameraId": "Cam-MGARR",
  "dateProcessed": "2022-04-11T06:40:10.063Z",
  "name": "AAC-2022-04-11T06:40:10.063Z",
  "MLModelId": "AAC-v1",
  "detectedBy": "GRN_Fog_Server"
}
```

SELD Raw Inference Result Data Model

Entity: SELD-Output

Global description:

A Data Model for the SELD inference model outputs of the MARVEL framework.

List of properties

List of properties defined as output by the inference model

- `startTime`: data type: float (seconds), absolute timestamp for the event start. Following the ISO8601 UTC format.
- `endTime`: data type: float (seconds), absolute timestamp for the event end. Following the ISO8601 UTC format.
- `label`: data type: string, value range defined by use case, for GRN4 ["motorcycle", "bicycle", "car", "bus"].
- `azimuthAngle`: data type: int (-180 - +180 degrees), azimuth of the sound source in relation of the mic array orientation. Zero value at the front. Azimuth angle is increasing counter-clockwise.
- `elevationAngle`: data type: int (-90 - +90 degrees), elevation of the sound source in relation of the mic array orientation. Zero value at the level of mic array.
- `azimuthVector`: data type: list of 2 array with 2 floats each, vector showing direction in GPS coordinates, format **lat1, lon1], [lat2, lon2**, (lat1,lon1) are the GPS coordinates of the mic array.

Extra properties to be passed by the inference model to relate to the original data stream and other MARVEL elements

- `cameraId`: Unique identifier of the camera from where the video was taken. Formatted according to the NGSI-LD standard (i.e. URN identifier). IDs for AV sources will be hard-coded manually in a configuration file or automatically assigned by the AV registry component and will include a 12-digit hexadecimal identifier.
- `dateProcessed`: Timestamp of the processing of the entity by the inference model.
- `description`: A description of this item
- `id`: Unique identifier of the entity. Formatted according to the NGSI standard. IDs for the especific media event can be automatically assigned by component.
- `MLModelId`: The id of the MLModel that generated this event. In this case it will be "SED-" and the version number of the model. It has to be one of the models defined in the registry.
- `name`: The name of this item. Typically in this case it could be the string "SED_" plus a timestamp of the moment of the execution
- `owner`: A field containing a JSON encoded sequence of characters referencing the unique Ids of the owner(s). In the case of MARVEL, we can use the pilot name and use case here. ENUM: [GRN3, GRN4, MT1, MT3, UNS1].
- `type`: The type of the entity. In this case "MediaEvent"
- `detectedBy`: Link to the device providing where the alert is detected (the link to the device id of the infrastructure in MARVEL)

Required properties by MARVEL

- `startTime`
- `endTime`
- `label`
- `id`

- `type`
- `owner`
- `cameraId`
- `dateProcessed`
- `MLModelId`
- `detectedBy`

Example payload

```
{
  "owner": "GRN4",
  "startTime": "2022-04-11T06:40:10.063Z",
  "endTime": "2022-04-11T06:40:10.063Z",
  "label": "bus",
  "azimuthAngle": 45,
  "elevationAngle": 0,
  "azimuthVector": "[40.741895, -73.989308],[40.741388868048034, -
73.98913840211132]",
  "id": "SELDEvent_2022-04-11T06:40:10.063Z",
  "type": "MediaEvent",
  "cameraId": "Cam-MGARR",
  "dateProcessed": "2022-04-11T06:40:10.063Z",
  "name": "SELD-2022-04-11T06:40:10.063Z",
  "MLModelId": "SELD-v1",
  "detectedBy": "GRN_Fog_Server"
}
```

VAD Raw Inference Result Data Model

Entity: VAD-Output

Global description:

A Data Model for the VAD inference model outputs of the MARVEL framework.

List of properties

List of properties defined as output by the inference model

- `startTime`: The start time of the event. Following the ISO8601 UTC format.
- `endTime`: The end time of the event. Following the ISO8601 UTC format.
- `category`: data type: string, a field containing the catefory of the acoustic event detected. ENUM: [speech, music] (currently only 2 values; new values could be added in future versions potentially).

Extra properties to be passed by the inference model to relate to the original data stream and other MARVEL elements

- `cameraId`: Unique identifier of the camera from where the video was taken. Formatted according to the NGSI standard (i.e. URN identifier). IDs for AV sources will be hard-coded manually in a configuration file or automatically assigned by the AV registry component and will include a 12-digit hexadecimal identifier.
- `dateProcessed`: Timestamp of the processing of the entity by the inference model.
- `description`: A description of this item
- `id`: Unique identifier of the entity. Formatted according to the NGSI standard. IDs for the especific media event can be automatically assigned by component.
- `MLModelId`: The id of the MLModel that generated this event. In this case it will be "AVAD-" and the version number of the model. It has to be one of the models defined in the registry.
- `name`: The name of this item. Typically in this case it could be the string "AVAD-" plus a timestamp of the moment of the execution
- `owner`: A field containing a JSON encoded sequence of characters referencing the unique Ids of the owner(s). In the case of MARVEL, we can use the pilot name and use case here. ENUM: [GRN3, GRN4, MT1, MT3, UNS1].
- `type`: The type of the entity. In this case "MediaEvent"
- `detectedBy`: Link to the device providing where the alert is detected (the link to the device id of the infrastructure in MARVEL)

Required properties by MARVEL

- `id`
- `type`
- `owner`
- `cameraId`
- `dateProcessed`
- `MLModelId`
- `startTime`
- `endTime`
- `detectedBy`

Example payload

```
{
  "id": "AVAD_Event_2022-04-11T06:40:10.063Z",
  "type": "MediaEvent",
  "cameraId": "Cam-MGARR",
  "dateProcessed": "2022-04-11T06:42:10.063Z",
  "name": "VAD-2022-04-11T06:40:10.063Z",
  "MLModelId": "AVAD-v1",
  "category": "speech",
  "startTime": "2022-04-01T10:07:59.618+01:58",
  "endTime": "2022-04-01T10:08:01.618+01:00",
  "detectedBy": "GRN_Fog_Server"
}
```

YOLO-SED Raw Inference Result Data Model

Entity: YOLO-SED-Output

Global description:

A Data Model for the YOLO-SED inference model outputs of the MARVEL framework.

List of properties

List of properties defined as output by the inference model

- `category`: data type: string, 'traffic'.
- `subCategory`: data type: string, a field containing the category of the anomaly detected. ENUM: ['pedestrian', 'bicycle'].
- `startTime`: data type: float (seconds), absolute timestamp for the event start. Following the ISO8601 UTC format.
- `endTime`: data type: float (seconds), absolute timestamp for the event end. Following the ISO8601 UTC format.

Extra properties to be passed by the inference model to relate to the original data stream and other MARVEL elements

- `cameraId`: Unique identifier of the camera from where the video was taken. Formatted according to the NGSI standard (i.e. URN identifier). IDs for AV sources will be hard-coded manually in a configuration file or automatically assigned by the AV registry component and will include a 12-digit hexadecimal identifier.
- `dateProcessed`: Timestamp of the processing of the entity by the inference model.
- `description`: A description of this item
- `id`: Unique identifier of the entity. Formatted according to the NGSI standard. IDs for the especific media event can be automatically assigned by component.
- `MLModelId`: The id of the MLModel that generated this event. In this case it will be "AVAD-" and the version number of the model. It has to be one of the models defined in the registry.
- `name`: The name of this item. Typically in this case it could be the string "AVAD-" plus a timestamp of the moment of the execution
- `owner`: A field containing a JSON encoded sequence of characters referencing the unique Ids of the owner(s). In the case of MARVEL, we can use the pilot name and use case here. ENUM: [GRN3, GRN4, MT1, MT3, UNS1].
- `type`: The type of the entity. In this case "Alert"
- `detectedBy`: Link to the device providing where the alert is detected (the link to the device id of the infrastructure in MARVEL)

Required properties by MARVEL

- `category`
- `subCategory`
- `id`
- `type`
- `owner`
- `cameraId`
- `dateProcessed`
- `MLModelId`
- `timestamp`
- `detectedBy`

Example payload

```
{
  "id": "YOLOSED_2022-04-11T06:40:10.063Z",
  "type": "Alert",
  "cameraId": "Cam-GRN-VA-01",
  "dateProcessed": "2022-04-11T06:42:10.063Z",
  "name": "YOLOSED-2022-04-11T06:40:10.063Z",
  "MLModelId": "YOLOSED-v1",
  "category": "traffic",
  "subCategory": "pedestrianOnRoad",
  "startTime": "2022-04-11T06:42:10.063Z",
  "endTime": "2022-04-11T06:42:10.063Z",
  "detectedBy": "GRN_EDGE_JETSON"
}
```

RBAD Raw Inference Result Data Model

Entity: RBAD-Output

Global description:

A Data Model for the RBAD inference model outputs of the MARVEL framework.

List of properties

List of properties defined as output by the inference model

- `predictedAnomaly`: data type: boolean, true or false, depending if the anomaly is detected (NOTE - Is this necessary? If the anomaly is not detected, it would suffice not to send any output).
- `anomalousProperty`: data type: string, a field containing the category of the anomaly detected.
  ENUM: ['bus_not_on_schedule', 'bicycle_not_on_path', 'large_veh_rush_hour', 'jaywalking'].
- `startTime`: data type: float (seconds), absolute timestamp for the event start. Following the ISO8601 UTC format.
- `endTime`: data type: float (seconds), absolute timestamp for the event end. Following the ISO8601 UTC format.

Extra properties to be passed by the inference model to relate to the original data stream and other MARVEL elements

- `cameraId`: Unique identifier of the camera from where the video was taken. Formatted according to the NGSI standard (i.e. URN identifier). IDs for AV sources will be hard-coded manually in a configuration file or automatically assigned by the AV registry component and will include a 12-digit hexadecimal identifier.
- `dateProcessed`: Timestamp of the processing of the entity by the inference model.
- `description`: A description of this item
- `id`: Unique identifier of the entity. Formatted according to the NGSI standard. IDs for the especific media event can be automatically assigned by component.
- `MLModelId`: The id of the MLModel that generated this event. In this case it will be "AVAD-" and the version number of the model. It has to be one of the models defined in the registry.
- `name`: The name of this item. Typically in this case it could be the string "AVAD-" plus a timestamp of the moment of the execution
- `owner`: A field containing a JSON encoded sequence of characters referencing the unique Ids of the owner(s). In the case of MARVEL, we can use the pilot name and use case here. ENUM: [GRN3, GRN4, MT1, MT3, UNS1].
- `type`: The type of the entity. In this case "Anomaly"
- `detectedBy`: Link to the device providing where the alert is detected (the link to the device id of the infrastructure in MARVEL)

Required properties by MARVEL

- `predictedAnomaly`
- `id`
- `type`
- `owner`
- `anomalousProperty`
- `cameraId`
- `dateProcessed`
- `MLModelId`

- `startTime`
- `endTime`
- `detectedBy`

Example payloads

```
{
  "predictedAnomaly": "true",
  "anomalousProperty": "jaywalking",
  "id": "RBAD_Event_2022-04-11T06:40:10.063Z",
  "type": "Anomaly",
  "cameraId": "Cam-GRN-VA-01",
  "dateProcessed": "2022-04-11T06:42:10.063Z",
  "name": "RBAD-2022-04-11T06:40:10.063Z",
  "MLModelId": "RBAD-v1",
  "startTime": 2022-04-11T06:40:10.063Z,
  "endTime":2022-04-11T06:40:14.063Z,
  "detectedBy": "GRN_Fog_Server"
}
```

GPURegex Raw Inference Result Data Model

Entity: GPURegex-Output

Global description:

A Data Model for the GPURegex output of the MARVEL framework.

List of properties

List of properties defined as output

- `startTime`: data type: float (seconds), absolute timestamp for the event start. Following the ISO8601 UTC format.
- `endTime`: data type: float (seconds), absolute timestamp for the event end. Following the ISO8601 UTC format.

Extra properties to be passed by the GPURegex to relate to the original data stream and other MARVEL elements

- `cameraId`: Unique identifier of the camera from where the video was taken. Formatted according to the NGSI-LD standard (i.e. URN identifier). IDs for AV sources will be hard-coded manually in a configuration file or automatically assigned by the AV registry component and will include a 12-digit hexadecimal identifier.
- `dateProcessed`: Timestamp of the processing of the entity by the inference model.
- `id`: Unique identifier of the entity. Formatted according to the NGSI standard. IDs for the especific media event can be automatically assigned by component.
- `MLModelId`: The id of the MLModel that generated this event. In this case it will be "GPUREGEX-" and the version number of the model. It has to be one of the models defined in the registry.
- `owner`: A field containing a JSON encoded sequence of characters referencing the unique Ids of the owner(s). In the case of MARVEL, we can use the pilot name and use case here. ENUM: [MT2, MT4].
- `type`: The type of the entity. In this case "Alert"
- `detectedBy`: Link to the device providing where the alert is detected (the link to the device id of the infrastructure in MARVEL)
- `alertSource`: Source of the alert ("GPURegex_MT_v01")
- `category`: Category of the Alert.
- `subCategory`: Describe the sub category of alert.

Required properties by MARVEL

- `startTime`
- `endTime`
- `id`
- `type`
- `owner`
- `cameraId`
- `dateProcessed`
- `MLModelId`
- `detectedBy`

Example payload

```
{
  "owner": "MT2",
  "startTime": "2022-04-11T06:40:10.063Z",
  "endTime": "2022-04-11T06:40:10.063Z",
  "id": "GPUREGEX_2022-04-11T06:40:10.063Z",
  "type": "Alert",
  "cameraId": "Cam-MGARR",
  "dateProcessed": "2022-04-11T06:40:10.063Z",
  "MLModelId": "GPURegex_MT_v01",
  "detectedBy": "MTFOG_2",
  "alertSource": "GPURegex_MT_v01",
  "category": "traffic",
  "subCategory": "carAccident"
}
```

Inference Result Verification Message Data Model

Inference Result Verification Message

These messages will be published from SmartViz to a Kafka topic in the DFB ("InferenceVerification") when a user reviews a specific inference result in SmartViz and provides feedback, i.e. determines if the result is verified or not. The DFB will obtain these messages and update the respective inference result entries stored in the DFB ElasticSearch accordingly. The DataCorpus will also subscribe to this topic to obtain these messages and store them so that they can be used for labeling in the context of future AI training purposes.

List of properties

- `index`: The index of ElasticSearch, under which the inference result has been stored.
- `inferenceResultId`: The id of the inference result about which the user has provided verification. The inference result type can be "MediaEvent", "Alert" or "Anomaly"
- `reviewed`: (boolean) Indicates whether the inference result has been reviewed by the user.
- `verified`: (boolean) Indicates whether the inference result is verified by the user or not.
- `verificationDate`: Date and time of verification of the inference result.

Example payload

```json
{
    "index": "tad",
    "inferenceResultId": "mediaEvent_1509702324600",
    "reviewed": true,
    "verified": true,
    "verificationDate": "2022-05-05T10:45:00Z"
}
```