Horizon 2020
European Union funding
for Research & Innovation

Big Data technologies and extreme-scale analytics

Multimodal Extreme Scale Data Analytics for Smart Cities Environments

D3.6: Efficient deployment of AI-optimised ML/DL models – final version[†]

Abstract: The objective of this deliverable is to provide an overview of the MARVEL Edge-to-Fog-Cloud framework, which serves as the deployment layer for the AI/DL MARVEL components. This framework encompasses the deployment logic that operates behind MARVdash, a proposed Kubernetes dashboard used to instantiate services as orchestrated containers and deploy them to desired execution sites, following an optimisation strategy. The primary aim of the optimisation strategy is to ensure that MARVEL components are deployed onto Kubernetes nodes based on their specific resource requirements and the available resources on the respective nodes. Furthermore, this document will outline methods for compressing machine learning algorithms/models by leveraging the resources present at the edge, such as reducing the size and operation time of deep learning models with millions of parameters. This compression approach can help minimise the computational overhead on edge servers.

Contractual Date of Delivery	30/06/2023
Actual Date of Delivery	30/06/2023
Deliverable Security Class	Public
Editor	*Manos Papoutsakis, Anthi Barmpaki, Emmanouil Michalodimitrakis (FORTH)*
Contributors	ATOS, AU, AUD, CNR, FBK, GRN, INTRA, ITML, STS, TAU, UNS, ZELUS
Quality Assurance	*Alessio Brutti (FBK)* *Goran Martic (UNS)*

DRAFT

[†] *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

DRAFT

Document Revisions & Quality Assurance

Internal Reviewers

1. *Alessio Brutti, (FBK)*
2. *Goran Martic, (UNS)*

Revisions

Version	Date	By	Overview
1.0	30/6/2023	Editors (FORTH)	Final version for submission to the EC
0.9	30/6/2023	Editors (FORTH)	Addressing comments from PC
0.8	28/06/2023	Editors (FORTH)	Final draft submitted to PC for quality check
0.7	28/06/2023	Editors (FORTH)	Approval from IRs
0.6	23/06/2023	Editors (FORTH)	2nd draft addressing comments from IRs
0.5	20/06/2023	Internal Reviewers (FBK, UNS)	Comments from Internal Reviewers
0.4	13/06/2023	Editors (FORTH), Contributors (ATOS, AU, AUD, CNR, FBK, GRN, INTRA, ITML, STS, TAU, UNS, ZELUS)	1st draft for internal review
0.3	08/05/2023	Editors (FORTH)	Revised ToC
0.2	02/05/2023	STPM	Comments on the TOC
0.1	08/05/2023	Editors (FORTH)	Initial ToC

DRAFT

Disclaimer

The work described in this document has been conducted within the MARVEL project. This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957337. This document does not reflect the opinion of the European Union, and the European Union is not responsible for any use that might be made of the information contained therein.
This document contains information that is proprietary to the MARVEL Consortium partners. Neither this document nor the information contained herein shall be used, duplicated or communicated by any means to any third party, in whole or in parts, except with prior written consent of the MARVEL Consortium.

Table of Contents

LIST OF TABLES 6
LIST OF LISTINGS 7
LIST OF FIGURES 8
LIST OF ABBREVIATIONS 9
EXECUTIVE SUMMARY 10
1 INTRODUCTION 11
1.1 PURPOSE AND SCOPE 11
1.2 CONTRIBUTION TO WP3 AND PROJECT OBJECTIVES 11
1.3 RELATION TO OTHER WORK PACKAGES, DELIVERABLES, AND ACTIVITIES 11
1.4 STRUCTURE OF THE REPORT 12
2 FINAL CONTAINER IMAGES OF MARVEL COMPONENTS 13
2.1 AI SUBSYSTEM 13
2.1.1 Visual Anomaly Detection – ViAD 13
2.1.2 Audio-Visual Anomaly Detection – AVAD 16
2.1.3 Visual Crowd Counting – VCC 16
2.1.4 Audio-Visual Crowd Counting – AVCC 19
2.1.5 Audio content analysis – AAC, SED, SELD, AT 19
2.1.6 Audio-Visual Vulnerable Road Users Detection: YOLO-SED 27
2.1.7 CATFlow 30
2.1.8 Text Anomaly Detection – TAD 30
2.1.9 Rule Based Anomaly Detection – RBAD 31
2.2 SECURITY, PRIVACY AND DATA PROTECTION SUBSYSTEM 33
2.2.1 EdgeSec Virtual Private Network – VPN 33
2.2.2 EdgeSec Trusted Execution Environment – TEE 33
2.2.3 VideoAnony 36
2.2.4 AudioAnony & VAD (devAIce) 36
2.3 DATA MANAGEMENT AND DISTRIBUTION SUBSYSTEM 37
2.3.1 StreamHandler 37
2.3.2 Data Fusion Bus – DFB 40
2.3.3 DatAna 41
2.3.4 Hierarchical Data Distribution – HDD 42
2.4 E2F2C SUBSYSTEM 43
2.4.1 GPURegex 43
2.4.2 DynHP 45
2.4.3 FedL 45
2.5 SYSTEM OUTPUTS SUBSYSTEM 45
2.5.1 SmartViz 45
2.5.2 MARVEL Data Corpus-as-a-Service 53
3 E2F2C OPTIMISED DEPLOYMENT SOLUTION 55
3.1 ARCHITECTURE OF THE MARVEL E2F2C FRAMEWORK 55
3.2 DEPLOYMENT OPTIMISATION 58
3.2.1 MARVdash new deployment method 58
3.2.2 Monitoring and optimisation tools 63
3.2.3 Real-time decision-making in all layers of the MARVEL E2F2C framework 68
4 MODEL OPTIMISATION FOR EFFICIENT INFERENCE 71
4.1 METHODS AND APPROACHES FOR EFFICIENT INFERENCE 71
4.1.1 Compression applied to Audio Visual Crowd Counting Models 71
4.1.2 Compression of Visual Crowd Counting Model 72

DRAFT

4.1.3 Deployment and evaluation of compression on an edge device 73
4.2 EFFICIENT ANOMALY DETECTION THROUGH DECENTRALISED AND UNSUPERVISED LEARNING 74
4.2.1 Motivation 74
4.2.2 Methodology description 75
4.2.3 Cost analysis for phase I 75
4.2.4 Cost analysis for phase II 76
4.2.5 Experimental setup 77
4.2.6 Results of Group Detection 77
4.2.7 Federated outlier detection 77
4.3 EFFICIENT FACE-SWAPPING USING HARDWARE-AWARE SCALING 78
4.3.1 Summary of the state-of-the-art 78
4.3.2 Description of work performed so far 78
4.3.3 Performance Evaluation 79

5 KPIS 80

5.1 PROJECT-RELATED KPIS 80
5.2 ASSET SPECIFIC KPIS 81

6 CONCLUSION 82

7 REFERENCES 83

DRAFT

List of Tables

Table 1: MARVEL components deployed through MARVdash 13
Table 2: MARVEL E2F2C Kubernetes nodes 56
Table 3: MARVdash API methods 69
Table 4: Output of GET services 69
Table 5: Output of GET templates 70
Table 6: Compression-accuracy performance comparison between compressed vs. uncompressed models trained on Disco Dataset 72
Table 7: Comparison between VCC full and compressed model 73
Table 8: Inference time performance evaluation 74
Table 9: Community detection 77
Table 10: AUC-ROC mean values summarising the performance of the proposed anomaly detection method 78
Table 11: Comparative analysis of the performance of the proposed approach 79

DRAFT

List of Listings

Listing 1: ViAD Dockerfile 16
Listing 2: VCC Dockerfile 18
Listing 3: Dockerfile for the audio AI component (AAC, SED, SELD, AT). 21
Listing 4: An example of the deployment template for SED component for edge layer of the infrastructure. 25
Listing 5: YOLO-SED Dockerfile 29
Listing 6: TAD container creation commands 30
Listing 7: RBAD Dockerfile 32
Listing 8: TEE & VideoAnony combined template YAML file 36
Listing 9: Service 2 Docker file for GRN Fog 2 deployment 38
Listing 10: Service 1 Docker file for GRN Fog 2 deployment 40
Listing 11: Use of NodePort method in the deployment YAML configuration file for the DFB Kafka service. 41
Listing 12: Exposing the Kafka topics to be monitored by the DataFusion connector service as parameters. 41
Listing 13: GPURegex container creation commands 43
Listing 14: GPURegex execution commands 43
Listing 15: Template used to deploy the newly added GPURegex image (Intel CPU and NVIDIA GPU) 45
Listing 16: SmartViz Dockerfiles and Docker-compose yaml 53

DRAFT

List of Figures

Figure 1. DatAna topologies for R2 ... 42
Figure 2. Topology of MARVEL E2F2C Kubernetes cluster ... 55
Figure 3. Create Service Updated ... 59
Figure 4. Create Service - Select layer ... 60
Figure 5. Create Service - Select Node ... 60
Figure 6. Create Service - Select GPU ... 61
Figure 7. Create Service - Without GPU ... 61
Figure 8. Create Service - GPU setting requests and limits ... 62
Figure 9. Create Service - Select Fog Layer ... 62
Figure 10. Create Service - Select Fog Layer Node with GPU ... 63
Figure 11. MARVdash updated menu ... 65
Figure 12. Grafana Dashboards ... 65
Figure 13. Compute Resources per namespace ... 66
Figure 14. Grafana MARVdash dashboard ... 66
Figure 15. Grafana MQTT dashboard ... 67
Figure 16. Zabbix MARVEL dashboard ... 68
Figure 17. Zabbix Data Corpus dashboard ... 68
Figure 18. SmartViz Service Management ... 70
Figure 19. Production of heatmaps ... 71
Figure 20. Example of A/V Input Output from Disco dataset ... 72
Figure 21. SASNet architecture [4] ... 73
Figure 22. Deployment procedure of AVCC on the Apollo Device ... 74
Figure 23. Block diagram of the proposed approach ... 78
Figure 24. Qualitative evaluation of the proposed method against the two state-of-the-art approaches 79

DRAFT

List of Abbreviations

AI	Artificial Intelligence
API	Application Programming Interface
AT	Audio Tagging
AVAD	Audio-Visual Anomaly Detection
AVCC	Audio-Visual Crowd Counting
CPU	Central Processing Unit
CRE	Container Runtime Environment
DFB	Data Fusion Bus
DL	Deep Learning
DMT	Decision-Making Toolkit
DNN	Deep Neural Network
E2F2C	Edge to Fog to Cloud
GA	Grant Agreement
GPU	Graphics Processing/Processor Unit
HDD	Hierarchical Data Distribution
HDFS	Hadoop Distributed Files System
HP	Hard Pruning
HTTP	Hypertext Transfer Protocol
ML	Machine Learning
MNIST	Modified National Institute of Standards and Technology
MQTT	Message Queuing Telemetry Transport
R2	2nd Release of the MARVEL Framework
REST	REpresentational State Transfer
SED	Sound Event Detection
TAD	Text Anomaly Detection
TEE	Trusted Execution Environment
UI	User Interface
VAD	Voice Activity Detection
VCC	Visual Crowd Counting
ViAD	Visual Anomaly Detection
VPN	Virtual Private Network
WP	Work Package
YAML	Ain't Markup Language

DRAFT

Executive Summary

The purpose of this deliverable is to provide the final version of the MARVEL Edge-to-Fog-to-Cloud (E2F2C) framework. The deliverable has been developed within the scope of WP3 of the MARVEL project under Grant Agreement (GA) No. 957337.

The document reports the outcomes of Tasks T3.4 and T3.5. As per the GA, the goals of T3.4 are to:

- Provide the deployment logic that will exploit the full potential of the personalised Federated Learning approach implemented in T3.2;
- Optimise and manage the deployment of Artificial Intelligence (AI) and Deep Learning (DL) components;
- Provide an optimisation strategy, for the component deployment, based on resource requirement and consumption.

Therefore, activities in the context of T3.4 were to:

- Create the infrastructure that will be used for the deployment of the MARVEL components. This infrastructure consists of a set of hosts part of a Kubernetes cluster. On top of this Kubernetes cluster, MARVdash was placed as a dashboard service for facilitating interaction with the underlying E2F2C testbed, by supplying the landing page for users, allowing them to launch services, design workflows, request resources, and specify other parameters related to execution through a user-friendly interface.
- Develop a deployment method, where MARVdash is used by the end users, allowing them to select execution environment for their applications without having to understand lower-level tools and interfaces.
- Install monitoring and managing tools such as Grafana, Prometheus, and Loki. Such tools allow for monitoring and analysing the performance and health of the underlying Kubernetes cluster.

The goals of T3.5 are to:

- Provide techniques and algorithms suitable for deployment at the edge layer;
- Study the compression requirements of AI/DL models based on resource availability;
- Compress such models for reduction of the computational overhead.

Therefore, the first activity of T3.5 was the exploration of different methods and approaches to improve inference efficiency and optimise models. Furthermore, the utilisation of decentralised and unsupervised learning for efficient anomaly detection was investigated and techniques for efficient face-swapping using hardware-aware scaling were presented.

DRAFT

1 Introduction

1.1 Purpose and scope

This document provides a comprehensive overview of the development process involved in creating an E2F2C execution environment, which includes the implementation of a specialised dashboard (MARVdash). This dashboard serves as a means to interact with the underlying environment (Kubernetes cluster), coordinate the execution of data management platforms and other software components, and handle external access to services that need to be accessible outside the MARVEL infrastructure.

Additionally, the document presents the methodology devised for compressing DL models during the training phase. It outlines the steps and techniques involved in compressing neural models and specifically focuses on the application of this methodology to the Audio-Visual Crowd Counting (AVCC) model developed by AU. The report provides insights into the practical implementation and results achieved through this compression approach.

1.2 Contribution to WP3 and project objectives

The deliverable is directly related to the achievement of MARVEL 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*".

One of the outputs of the above objective is to have a distributed E2F2C deep learning approach including a model optimisation approach that adapts to the E2F2C resources that is related to Task 3.4: *Adaptive E2F2C distribution and optimisation of AI tasks* and Task 3.5: *Edge-optimal ML/DL deployment for multimodal processing*. The deliverable constitutes the output of those tasks.

The work conducted in the above tasks, described in this deliverable, contributes mainly in two objectives of WP3:

- (iv) *optimise ML deployment in the E2F2C infrastructure in a continuous manner*
- (v) *deploy ML algorithms at all layers of E2F2C.*

These objectives are fulfilled with:

- The new deployment method offered by MARVdash.
- The monitoring tools that are deployed i.e. Prometheus, Grafana, Loki and Zabbix.
- The scheduler utilised by Kubernetes that follows the best fit approach.
- The API offered by MARVdash for initializing and stopping services at any layer.

Finally, the work done under T3.4 and T3.5 contributes to achieving the KPIs mentioned in Section 5.

1.3 Relation to other work packages, deliverables, and activities

This document has strong connections to various tasks within WP3. T3.1 focuses on developing AI-based methods for data privacy, T3.2 aims to create a Federated Learning (FL) framework based on distributed data, and T3.3 addresses the development of multimodal audio-visual AI models. The deployment logic discussed in this document enables the realisation of the

DRAFT

personalised Federated Learning approach implemented in T3.2, as well as embodies the E2F2C approach in general for all components.

Furthermore, the Kubernetes dashboard for the MARVEL E2F2C framework will play a crucial role in deploying all the other MARVEL components, which are responsible for functionalities beyond AI model training and inference. These components are involved in data transfer and management. As a result, this document relates to other tasks across different Work Packages (WPs) dedicated to the development of MARVEL components. It also connects with the corresponding deliverables that describe the functionality of these components.

Additionally, this document is highly pertinent to the tasks within all technical WPs, namely WP2, WP3, WP4, and WP5, as it aligns with the overarching objective of delivering the MARVEL E2F2C framework. Spanning across multiple WPs, the contents of this document serve as a crucial foundation for the successful implementation and realization of the MARVEL E2F2C framework.

1.4 Structure of the report

The deliverable focuses on the progress performed from the previous version, that is D3.2[1]. This approach is followed throughout the subsequent sections. In cases where we deemed it essential for the completeness of the deliverable, we opted to provide a summary of background information that was already addressed in the initial version.

The deliverable is structured as follows. Section 2 provides a comprehensive overview of all the MARVEL components that utilise the MARVdash dashboard for deployment on one or more nodes within the MARVEL E2F2C framework.

In Section 3, the focus shifts to the proposed deployment solution. It starts with a description of the MARVEL E2F2C framework, which consists of two main components, i.e., the underlying Kubernetes cluster and the MARVdash dashboard on top of it. This section also delves into the selected monitoring and managing tools that report the performance and health of the cluster, and the provided MARVdash Application Programming Interface (API) that allows for the real-time decision-making functionality offered by the MARVEL E2F2C framework.

Section 4 focuses on the exploration of diverse approaches to optimise models, improve inference efficiency, enable efficient anomaly detection, and enhance face-swapping techniques through the implementation of hardware-aware strategies. The compression methods, namely DynHP and AVCC, are described in detail.

Section 5 focuses on the description of Key Performance Indicators (KPIs) and evaluates the extent to which they have been met based on the progress made until M30 of the project's lifespan.

Finally, Section 6 summarises the conclusions drawn from the deliverable, providing a concise overview of the document's key findings.

DRAFT

[1] MARVEL D3.2 - Efficient deployment of AI-optimised ML/DL models – initial version, 2022. https://doi.org/10.5281/zenodo.6821232

2 Final container images of MARVEL components

The foundation of the developed MARVEL E2F2C framework relies on a Kubernetes cluster, enabling the deployment of various MARVEL components across its nodes. By leveraging containerisation, the deployment process within the Kubernetes creates individual isolated fully packaged and portable computing environments. In this section, we outline the methodology employed to construct these execution environments for each specific MARVEL component, utilising Docker containers.

A summary table with all the MARVEL components that are deployed in the MARVEL E2F2C framework (Kubernetes cluster) is depicted below (Table 1). Details regarding the functionality of each component and the way they are deployed can be found in the next subsections.

Table 1: MARVEL components deployed through MARVdash

Component name	Component Owner	E2F2C Layer
Visual anomaly detection (ViAD)	AU	Cloud
Audio-Visual anomaly detection (AVAD)	AU	Fog, Cloud
Visual crowd counting (VCC)	AU	Fog, Cloud
Audio-Visual crowd counting (AVCC)	AU	Cloud
Automated audio captioning (AAC)	TAU	Fog
Sound event detection (SED)	TAU	Edge, Fog
Sound event localisation and detection (SELD)	TAU	Edge
Audio tagging (AT)	TAU	Fog, Cloud
Audio-Visual Vulnerable Road Users Detection (YOLO-SED)	AU	Edge
CATFlow	GRN	Edge, Fog
Text Anomaly Detection (TAD)	GRN	Edge, Fog
Rule Based Anomaly Detection (RBAD)	AU	Edge, Fog
EdgeSec TEE	FORTH	Edge
VideoAnony	FBK	Edge, Fog, Cloud
AudioAnony & VAD (devAIce)	FBK	Edge, Cloud
StreamHandler	INTRA	Fog
Data Fusion Bus (DFB)	ITML	Cloud
DatAna	ATOS	Edge, Fog, Cloud
Hierarchical Data Distribution (HDD)	CNR	Cloud
GPURegex	FORTH	Fog
DynHP	CNR	Cloud
FedL	UNS	Edge, Fog
SmartViz	ZELUS	Cloud

2.1 AI subsystem

2.1.1 Visual Anomaly Detection – ViAD

The ViAD component learns to differentiate between normal and anomalous situations in visual-only videos. Therefore, an anomaly can be any novel event that has not occurred in the scene before, but those that have been present in the training dataset are detected with higher

DRAFT

accuracy. It receives as an input a sequence of video frames and it produces a single flag specifying whether any anomalies are present in the input sequence.

The base image for the ViAD component is the Pytorch:1.12.0 container (Listing 1). The next commands set the time zone, required to properly synchronise with the other MARVEL components. The RUN command installs prerequisite software. Then, the model, python requirements as well as the Mudas library, developed by AU for the MARVEL project, are copied and installed within the image. Finally, the last command starts the component, passing the required parameters.

```
FROM pytorch/pytorch:1.12.0-cuda11.3-cudnn8-runtime
ARG DEBIAN_FRONTEND=noninteractive
ENV TZ=Europe/Copenhagen
RUN apt-get update && \
    apt-get install -y build-essential && \
    apt-get install -y libgl1 libsndfile1 && \
    apt-get install -y ffmpeg && \
    apt-get clean
COPY containers/container_source/vggish_pretrained.pth /app/
COPY containers/container_source/requirements.txt /tmp/
RUN pip install --default-timeout=100 -r /tmp/requirements.txt
COPY containers/container_source/mudas-0.3.8-py3-none-any.whl /tmp/
RUN pip install --default-timeout=100 /tmp/mudas-0.3.8-py3-none-any.whl --no-
dependencies
WORKDIR /app
COPY containers/container_source/utils.py /app/
COPY containers/container_source/model_repo.py /app/
COPY containers/container_source/preprocessing_core.py /app/
COPY containers/container_source/pp_source_avad_viad.py /app/
CMD ["sh", "-c", "echo PARAMETERS:; \
    echo ------; \
    echo ACCESS_AV_REGISTRY=${ACCESS_AV_REGISTRY};\
    echo ANOMALY_THRESHOLD=${ANOMALY_THRESHOLD};\
    echo AUDIO_LENGTH=${AUDIO_LENGTH};\
    echo AUDIO_SIZE=${AUDIO_SIZE};\
    echo AUDIO_URL=${AUDIO_URL};\
    echo BROKER=${BROKER};\
    echo CAMERA_ID=${CAMERA_ID};\
    echo CROP_PARAMS=${CROP_PARAMS};\
    echo DETECTED_BY=${DETECTED_BY};\
    echo EVENT_TYPE=${EVENT_TYPE};\
    echo EXAMPLE_HOP_SECS=${EXAMPLE_HOP_SECS};\
    echo EXAMPLE_WINDOW_LENGTH_SECS=${EXAMPLE_WINDOW_LENGTH_SECS};\
    echo GET_MODEL_REPO=${GET_MODEL_REPO};\
    echo ID=${ID};\
    echo LIMIT=${LIMIT};\
    echo LOG_OFFSET=${LOG_OFFSET};\
    echo LOWER_EDGE_HERTZ=${LOWER_EDGE_HERTZ};\
```

```
    echo MODE=${MODE};\
    echo MODEL_NAME=${MODEL_NAME};\
    echo MQTT_PASSWORD=${MQTT_PASSWORD};\
    echo MQTT_USERNAME=${MQTT_USERNAME};\
    echo NUM_MEL_BINS=${NUM_MEL_BINS};\
    echo OWNER=${OWNER};\
    echo OWNER_BIN=${OWNER_BIN};\
    echo PORT=${PORT};\
    echo REGISTRY_URL=${REGISTRY_URL};\
    echo SECOND_CAMERA_ID=${SECOND_CAMERA_ID};\
    echo SFT_HOP_LENGTH_SECS=${SFT_HOP_LENGTH_SECS};\
    echo SFT_WINDOW_LENGTH_SECS=${SFT_WINDOW_LENGTH_SECS};\
    echo TARGET_SAMPLE_RATE=${TARGET_SAMPLE_RATE};\
    echo TOPIC=${TOPIC};\
    echo VERBOSE=${VERBOSE};\
    echo SHORT_SIDE=${SHORT_SIDE};\
    echo VIDEO_URL=${VIDEO_URL};\
    echo SEQUENCE_LENGTH=${SEQUENCE_LENGTH};\
    echo USE_GPU=${USE_GPU};\
    echo WORKER_COUNT=${WORKER_COUNT};\
    echo QUEUE_SIZE=${QUEUE_SIZE};\
    echo AUDIO_CONTEXT_LENGTH=${AUDIO_CONTEXT_LENGTH};\
    python pp_source_avad_viad.py \
        --access_av_registry ${ACCESS_AV_REGISTRY}  \
        --anomaly_threshold ${ANOMALY_THRESHOLD}  \
        --audio_length ${AUDIO_LENGTH}  \
        --audio_size ${AUDIO_SIZE} \
        --audio_url ${AUDIO_URL} \
        --broker ${BROKER} \
        --camera_id ${CAMERA_ID} \
        --crop_params ${CROP_PARAMS} \
        --detected_by ${DETECTED_BY} \
        --event_type ${EVENT_TYPE} \
        --example_hop_secs ${EXAMPLE_HOP_SECS} \
        --example_window_length_secs ${EXAMPLE_WINDOW_LENGTH_SECS} \
        --get_model_repo ${GET_MODEL_REPO} \
        --id ${ID} \
        --limit ${LIMIT} \
        --log_offset ${LOG_OFFSET} \
        --lower_edge_hertz ${LOWER_EDGE_HERTZ} \
        --mode ${MODE}  \
        --model_name ${MODEL_NAME} \
        --mqtt_password ${MQTT_PASSWORD} \
        --mqtt_username ${MQTT_USERNAME} \
        --num_mel_bins ${NUM_MEL_BINS} \
        --owner ${OWNER} \
        --owner_bin ${OWNER_BIN} \
```

DRAFT

```
        --port ${PORT} \
        --registry_url ${REGISTRY_URL} \
        --second_camera_id ${SECOND_CAMERA_ID} \
        --sft_hop_length_secs ${SFT_HOP_LENGTH_SECS} \
        --sft_window_length_secs ${SFT_WINDOW_LENGTH_SECS} \
        --sft_window_length_secs ${SFT_WINDOW_LENGTH_SECS} \
        --topic ${TOPIC} \
        --verbose ${VERBOSE} \
        --short_side ${SHORT_SIDE} \
        --video_url ${VIDEO_URL} \
        --sequence_length ${SEQUENCE_LENGTH} \
        --worker_count ${WORKER_COUNT} \
        --queue_sizes ${QUEUE_SIZE} \
        --audio_context_length ${AUDIO_CONTEXT_LENGTH} \
        --use_gpu ${USE_GPU}"]
```

Listing 1: ViAD Dockerfile

2.1.2 Audio-Visual Anomaly Detection – AVAD

The ViAD component learns to differentiate between normal and anomalous situations in audio-visual videos. Therefore, an anomaly can be any novel event that has not occurred in the scene before, but those that have been present in the training dataset are detected with higher accuracy. It receives as an input a sequence of video frames and it produces a single flag specifying whether any anomalies are present in the input sequence.

The structure of the AVAD Dockerfile is the same as that of the ViAD component presented in Section 2.1.1.

2.1.3 Visual Crowd Counting – VCC

The VCC component counts the total number of people present in an image. Since the annotations in the training data specify the locations of the heads, crowd counting can be viewed as counting the total number of heads present in the image. The input to this component is an image from a scene that may contain people, and the output is a number representing the total count of people in that scene. Optionally, the output may contain a heatmap specifying the density of people for each pixel of the image (also known as “density map”).

The VCC component’s Dockerfile follows the same structure as those for ViAD and AVAD (Listing 2). Pytorch:1.12.0 is the base image. The next commands copy the required files and install the libraries. The last command starts the VCC component loop.

```
FROM pytorch/pytorch:1.12.0-cuda11.3-cudnn8-runtime

ARG DEBIAN_FRONTEND=noninteractive
ENV TZ=Europe/Copenhagen
RUN apt-get update && \
    apt-get install -y build-essential && \
    apt-get install -y libgl1 libsndfile1 && \
    apt-get install -y ffmpeg && \
    apt-get clean
```

DRAFT

```
COPY containers/container_source/vggish_pretrained.pth /app/
COPY containers/container_source/requirements.txt /tmp/
RUN pip install --default-timeout=100 -r /tmp/requirements.txt
COPY containers/container_source/mudas-0.3.9-py3-none-any.whl /tmp/
RUN pip install --default-timeout=100 /tmp/mudas-0.3.9-py3-none-any.whl --no-
dependencies
WORKDIR /app
COPY containers/container_source/utils.py /app/
COPY containers/container_source/model_repo.py /app/
COPY containers/container_source/pp_source_avcc_vcc.py /app/
CMD ["sh", "-c", "echo PARAMETERS:; \
    echo ------; \
    echo ACCESS_AV_REGISTRY=${ACCESS_AV_REGISTRY};\
    echo AUDIO_CONTEXT_LENGTH=${AUDIO_CONTEXT_LENGTH};\
    echo AUDIO_LENGTH=${AUDIO_LENGTH};\
    echo AUDIO_SIZE=${AUDIO_SIZE};\
    echo AUDIO_URL=${AUDIO_URL};\
    echo BROKER=${BROKER};\
    echo CAMERA_ID=${CAMERA_ID};\
    echo CROP_PARAMS=${CROP_PARAMS};\
    echo DETECTED_BY=${DETECTED_BY};\
    echo EVENT_TYPE=${EVENT_TYPE};\
    echo EXAMPLE_HOP_SECS=${EXAMPLE_HOP_SECS};\
    echo EXAMPLE_WINDOW_LENGTH_SECS=${EXAMPLE_WINDOW_LENGTH_SECS};\
    echo GET_MODEL_REPO=${GET_MODEL_REPO};\
    echo HEATMAP_FREQUENCY=${HEATMAP_FREQUENCY};\
    echo ID=${ID};\
    echo LIMIT=${LIMIT};\
    echo LOG_OFFSET=${LOG_OFFSET};\
    echo LOWER_EDGE_HERTZ=${LOWER_EDGE_HERTZ};\
    echo MODE=${MODE};\
    echo MODEL_NAME=${MODEL_NAME};\
    echo MQTT_PASSWORD=${MQTT_PASSWORD};\
    echo MQTT_PORT=${MQTT_PORT};\
    echo MQTT_USERNAME=${MQTT_USERNAME};\
    echo NUM_MEL_BINS=${NUM_MEL_BINS};\
    echo OWNER=${OWNER};\
    echo OWNER_BIN=${OWNER_BIN};\
    echo QUEUE_SIZES=${QUEUE_SIZES};\
    echo REGISTRY_URL=${REGISTRY_URL};\
    echo SECOND_CAMERA_ID=${SECOND_CAMERA_ID};\
    echo SFT_HOP_LENGTH_SECS=${SFT_HOP_LENGTH_SECS};\
    echo SFT_WINDOW_LENGTH_SECS=${SFT_WINDOW_LENGTH_SECS};\
    echo TARGET_SAMPLE_RATE=${TARGET_SAMPLE_RATE};\
    echo TIMESTAMP_MODE=${TIMESTAMP_MODE};\
    echo TOPIC=${TOPIC};\
    echo USE_GPU=${USE_GPU};\
```

```
    echo VERBOSE=${VERBOSE};\
    echo VIDEO_SIZE=${VIDEO_SIZE};\
    echo VIDEO_URL=${VIDEO_URL};\
    echo WORKER_COUNT=${WORKER_COUNT};\
    python pp_source_avcc_vcc.py \
        --access_av_registry ${ACCESS_AV_REGISTRY}  \
        --audio_context_length ${AUDIO_CONTEXT_LENGTH} \
        --audio_length ${AUDIO_LENGTH}  \
        --audio_size ${AUDIO_SIZE} \
        --audio_url ${AUDIO_URL} \
        --broker ${BROKER} \
        --camera_id ${CAMERA_ID} \
        --crop_params ${CROP_PARAMS} \
        --detected_by ${DETECTED_BY} \
        --event_type ${EVENT_TYPE} \
        --example_hop_secs ${EXAMPLE_HOP_SECS} \
        --example_window_length_secs ${EXAMPLE_WINDOW_LENGTH_SECS} \
        --get_model_repo ${GET_MODEL_REPO} \
        --heatmap_frequency ${HEATMAP_FREQUENCY} \
        --id ${ID} \
        --limit ${LIMIT} \
        --log_offset ${LOG_OFFSET} \
        --lower_edge_hertz ${LOWER_EDGE_HERTZ} \
        --mode ${MODE}  \
        --model_name ${MODEL_NAME} \
        --mqtt_password ${MQTT_PASSWORD} \
        --mqtt_port ${MQTT_PORT} \
        --mqtt_username ${MQTT_USERNAME} \
        --num_mel_bins ${NUM_MEL_BINS} \
        --owner ${OWNER} \
        --owner_bin ${OWNER_BIN} \
        --queue_sizes ${QUEUE_SIZES} \
        --registry_url ${REGISTRY_URL} \
        --second_camera_id ${SECOND_CAMERA_ID} \
        --sft_hop_length_secs ${SFT_HOP_LENGTH_SECS} \
        --sft_window_length_secs ${SFT_WINDOW_LENGTH_SECS} \
        --timestamp_mode ${TIMESTAMP_MODE} \
        --topic ${TOPIC} \
        --use_gpu ${USE_GPU} \
        --verbose ${VERBOSE} \
        --video_url ${VIDEO_URL} \
        --worker_count ${WORKER_COUNT} \
        "]
```

Listing 2: VCC Dockerfile

2.1.4 Audio-Visual Crowd Counting – AVCC

The AVCC component counts the total number of people present in an image. Since the annotations in the training data specify the locations of the heads, crowd counting can be viewed as counting the total number of heads present in the image. The input to this component is an image from a scene that may contain people and a 1-second audio snippet preceding that frame. The output is a number representing the total count of people in that scene. Optionally, the output may contain a heatmap specifying the density of people for each pixel of the image (also known as "density map").

The AVCC component's Dockerfile follows the same structure as that for VCC (Listing 2).

2.1.5 Audio content analysis – AAC, SED, SELD, AT

Four audio content analysis components released by TAU, automated audio captioning (AAC), sound event detection (SED), sound event localisation and detection (SELD), and audio tagging (AT), share the same code base. The component type is selected at the component deployment and the internal data processing pipeline is selected accordingly and the correct AI model is fetched from the AI Model Repository component. The component can be run either in the cloud, in the edge or in the fog layer of the infrastructure. Nodes in these layers can have a high-performance Central Processing Unit (CPU) or a Graphics Processing/Processor Unit (GPU) available for AI computations. The component is designed to consume continuous audio-visual streams and operate in real-time.

The component is deployed with a Docker Image shown in Listing 3. The component is implemented in Python and the audio AI subsystem is using PyTorch machine learning framework. The environmental variables inside the Docker container are used to relay parameters from the Kubernetes deployment template at the deployment time to the component launch command inside the Docker image.

The component is deployed into MARVEL infrastructure using the Kubernetes deployment template in YAML format. An example of such template is shown in Listing 4. This example has default parameters set to deploy the SED component for the GRN2 use case (Road User Behaviour) in the edge layer of the infrastructure. The same template can be used to deploy other types of components by changing the "COMPONENT" parameter (sets the component type, e.g., SED, AT, SELD) and the "MODEL_ID" parameter (sets the used AI model). The audio-visual stream source information is fetched at the component start-up from the AV Registry component based on the given "CAMERAID" parameter. Similarly, the AI model is fetched at component start-up from the Model Repository component based on the given "MODEL ID" parameter.

DRAFT

```
FROM python:3.9-bullseye
ENV PYTHONDONTWRITEBYTECODE=1
ENV PYTHONUNBUFFERED=1
RUN apt-get update -y && apt-get install -y --no-install-recommends build-
essential gcc libsndfile1
RUN apt-get update -y && apt-get upgrade -y && apt-get install -y ffmpeg

WORKDIR /app
COPY ai.py /app/
COPY base_process.py /app/
COPY daemon.py /app/
```

```
COPY receiver.py /app/
COPY transmitter.py /app/
COPY utils.py /app/
COPY models.py /app/
COPY config/ /app/config/

RUN adduser -u 5678 --disabled-password --gecos "" appuser && chown -R appuser
/app
USER appuser
ENV PATH="/home/appuser/.local/bin:$PATH"

RUN python -m pip install numpy==1.20.3
RUN python -m pip install torch==1.9.1+cu111 -f
https://nelsonliu.me/files/pytorch/whl/torch_stable.html
RUN python -m pip install dcase_util==0.2.19
RUN python -m pip install ffmpeg-python==0.2.0
RUN python -m pip install multiprocessing-logging==0.3.1
RUN python -m pip install paho-mqtt==1.6.1
RUN python -m pip install pika==1.2.0
RUN python -m pip install minio==7.1.8
RUN python -m pip install transformers==4.21.2
RUN python -m pip install essential_generators==1.0
RUN python -m pip install logger_tt==1.7.3

CMD ["sh", "-c", "\
    python daemon.py \
    --component ${COMPONENT}  \
    --owner=${OWNER} \
    --infrastructure_id ${INFRASTRUCTURE_ID} \
    --avr_hostname ${AVR_HOSTNAME} \
    --avr_port ${AVR_PORT} \
    --registry_id ${CAMERAID} \
    --model_url ${MODEL_URL} \
    --model_id ${MODEL_ID} \
    --mqtt_enabled ${MQTT_ENABLED} \
    --mqtt_hostname ${MQTT_HOSTNAME} \
    --mqtt_port ${MQTT_PORT} \
    --mqtt_topic ${MQTT_TOPIC} \
    --generate_input ${GENERATE_INPUT} \
    --generate_output ${GENERATE_OUTPUT} \
    --local ${LOCAL} \
    --use_local_model ${LOCAL_MODEL}  \
    --use_local_avregistry ${LOCAL_AVREGISTRY} \
    --show_config ${SHOW_CONFIG} \
    --show_input ${SHOW_INPUT} \
    --show_output ${SHOW_OUTPUT} \
    --input_stream ${INPUT_STREAM} \
```

```
    --device ${DEVICE} \
    "]
```

Listing 3: Dockerfile for the audio AI component (AAC, SED, SELD, AT).

```yaml
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: $NAME
spec:
  rules:
  - host: $HOSTNAME
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: $NAME
            port:
              number: 8080
---
apiVersion: v1
kind: Service
metadata:
  name: $NAME
spec:
  type: ClusterIP
  ports:
  - port: 8080
  selector:
    app: $NAME
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: $NAME
spec:
  replicas: 1
  selector:
    matchLabels:
      app: $NAME
  template:
    metadata:
      labels:
        app: $NAME
    spec:
```

```
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
            - matchExpressions:
              - key: Layer
                operator: In
                values:
                - GRNEDGE1
      containers:
      - name: $NAME
        image: 192.168.50.1:5000/tau_audio_ai:1.9.62
        ports:
        - containerPort: 8080
        env:
        - name: COMPONENT
          value: $COMPONENT
        - name: DEVICE
          value: $DEVICE
        - name: OWNER
          value: $OWNER
        - name: AVR_HOSTNAME
          value: $AVR_HOSTNAME
        - name: AVR_PORT
          value: $AVR_PORT
        - name: CAMERAID
          value: $CAMERAID
        - name: INFRASTRUCTURE_ID
          value: $INFRASTRUCTURE_ID
        - name: MODEL_URL
          value: $MODEL_URL
        - name: MODEL_ID
          value: $MODEL_ID
        - name: MQTT_ENABLED
          value: $MQTT_ENABLED
        - name: MQTT_HOSTNAME
          value: $MQTT_HOSTNAME
        - name: MQTT_PORT
          value: $MQTT_PORT
        - name: MQTT_TOPIC
          value: $MQTT_TOPIC
        - name: GENERATE_INPUT
          value: $GENERATE_INPUT
        - name: GENERATE_OUTPUT
          value: $GENERATE_OUTPUT
        - name: LOCAL
          value: $LOCAL
```

```
        - name: LOCAL_MODEL
          value: $LOCAL_MODEL
        - name: LOCAL_AVREGISTRY
          value: $LOCAL_AVREGISTRY
        - name: SHOW_CONFIG
          value: $SHOW_CONFIG
        - name: SHOW_INPUT
          value: $SHOW_INPUT
        - name: SHOW_OUTPUT
          value: $SHOW_OUTPUT
        - name: INPUT_STREAM
          value: $INPUT_STREAM
        resources:
          requests:
            memory: "3G"
            cpu: "1"
          limits:
            aliyun.com/gpu-mem: 1
            memory: $MEMORY
            cpu: $CPUS
      tolerations:
      - key: "Layer"
        operator: "Equal"
        value: "GRNEDGE1"
---
kind: Template
name: TAU Audio AI / GRNEDGE1 / GPU
description: TAU Audio AI / GRN2 / GRNEDGE1 / GPU / SED
variables:
- name: NAME
  default: tau-audio-ai-grnedge1-sed

- name: HOSTNAME
  default: tau-sed.example.com

- name: OWNER
  default: GRN2
  help: Owner override

- name: COMPONENT
  default: SED
  help: Component type [SED, AT, AAC, SELD]

- name: DEVICE
  default: cuda
  help: Set device to cpu or cuda
```

```
- name: AVR_HOSTNAME
  default: avregistry-grn.karvdash-tkanellos
  help: AV Registry hostname

- name: AVR_PORT
  default: "\"3000\""
  help: AV Registry port

- name: CAMERAID
  default: "\"Cam-GRN-VA-01-Audio\""
  help: Registry id to query AV registry

- name: INFRASTRUCTURE_ID
  default: GRNEDGE1
  help: Infrastructure ID

- name: MODEL_URL
  default: minio.karvdash-minio.svc:9000
  help: Model Registry URL

- name: MODEL_ID
  default: SED_GRN2_v01
  help: AI model ID

- name: MQTT_ENABLED
  default: "\"1\""
  help: MQTT output enabled, possible values are "0" or "1", make sure to use
quotes.

- name: MQTT_HOSTNAME
  default: mqtt-kubernetesgrnedge1.karvdash-datanagrnedge.svc
  help: MQTT broker hostname

- name: MQTT_PORT
  default: "\"1883\""
  help: MQTT broker port

- name: MQTT_TOPIC
  default: SED
  help: MQTT topic [SED, AT]

- name: GENERATE_INPUT
  default: "\"0\""
  help: Generate random input signal for debugging purposes, possible values are
"0" or "1", make sure to use quotes.

- name: GENERATE_OUTPUT
```

```
  default: "\"0\""
  help: Generate random output for debugging purposes, possible values are "0"
or "1", make sure to use quotes.

- name: LOCAL
  default: "\"0\""
  help: Run component in local mode, av registry and AI model loaded from local
registry, possible values are "0" or "1", make sure to use quotes.

- name: LOCAL_MODEL
  default: "\"0\""
  help: Run component with local MLMODEL, possible values are "0" or "1", make
sure to use quotes.

- name: LOCAL_AVREGISTRY
  default: "\"0\""
  help: Run component with local AV Registry, possible values are "0" or "1",
make sure to use quotes.

- name: SHOW_CONFIG
  default: "\"1\""
  help: Show config information, possible values are "0" or "1", make sure to
use quotes.

- name: SHOW_INPUT
  default: "\"1\""
  help: Show input, possible values are "0" or "1", make sure to use quotes.

- name: SHOW_OUTPUT
  default: "\"1\""
  help: Show output, possible values are "0" or "1", make sure to use quotes.

- name: INPUT_STREAM
  default: "\"\""
  help: Input stream URL override

- name: MEMORY
  default: "\"6G\""
  help: Limit for memory usage (e.g. "6G")

- name: CPUS
  default: "\"6\""
  help: Limit for CPU usage (e.g. "6")
```

Listing 4: An example of the deployment template for SED component for edge layer of the infrastructure.

Automated audio captioning – AAC

The automated audio captioning (AAC) component creates textual descriptions with full sentences for an 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 direct descriptions for humans accessing audio-visual streams, as well as for further text-based analysis to assist the decision-making process. The AAC component is used in monitoring use cases in MARVEL to provide descriptive captions of audio-visual segments accessed by the monitoring system users.

The AAC component is deployed by specialising in the base Audio AI component by setting the “COMPONENT” parameter to “AAC” at the deployment and setting the appropriate “MODELID” to fetch the AAC model from the AI Model Repository component.

Sound event detection – SED

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. This functionality is used in the various use cases of MARVEL to offer the ability to detect actions and events through sound. 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 a deeper understanding of the scene. The specific sound events detected by the component will be dependent on the use cases. A SED component takes as an input an audio signal and provides detection of sound events in pre-specified units of time.

The SED component is deployed by specialising in the base Audio AI component by setting the “COMPONENT” parameter to “SED” at the deployment and setting the appropriate “MODELID” to fetch the SED model from the AI Model Repository component.

Sound event localisation and detection – SELD

The sound event localisation and detection (SELD) task is a joint task where a system 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 associated with an event. The localisation is performed concerning the microphone position and orientation. A SELD system is used in monitoring use cases in MARVEL to produce an in-depth view of the auditory scene in use cases where sound localisation is an important aspect.

The SELD component is deployed by specialising in the base Audio AI component by setting the “COMPONENT” parameter to “SELD” at the deployment and setting the appropriate “MODELID” to fetch the SELD model from the AI Model Repository component.

Audio tagging - AT

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 depend on the use case specifications. The information about the sound class activity can be used as standalone information or as complementary information to other systems to gain a deeper understanding of the scene.

DRAFT

The AT component is deployed by specialising in the base Audio AI component by setting the "COMPONENT" parameter to "AT" at the deployment and setting the appropriate "MODELID" to fetch AT model from the AI Model Repository component.

2.1.6 Audio-Visual Vulnerable Road Users Detection: YOLO-SED

The YOLO-SED component analyses audio-visual data on the edge and detects anomalies using the YOLO object detector and SED audio analysis module. The component receives input images from a scene, coupled with audio data. The outputs of YOLO and SED modules are fused with a rule-based approach to detect if there is an anomaly in the current time frame or not, represented by a single Boolean value.

YOLO-SED component can be deployed directly on the device, or in a containerised form using Docker. The Docker image is based on a pre-built image for NVIDIA Jetson devices that is compatible with the 4.6 JetPack: nvcr.io/nvidia/l4t-pytorch:r32.7.1-pth1.10-py3.

Dockerfile sets locale parameters to avoid Python ASCII error, removes inaccessible links from the sources list, copies needed files and installs system and Python requirements.

The container relies on NVidia runtime, which should be either set as a default runtime on the Jetson device or be specifically called during the docker run command. This allows accessing GPU libraries from inside the Docker container.

```
FROM nvcr.io/nvidia/l4t-pytorch:r32.7.1-pth1.10-py3

ARG DEBIAN_FRONTEND=noninteractive
ENV TZ=Europe/Copenhagen
ENV export LC_ALL=en_US.UTF-8
ENV export LANG=en_US.UTF-8

# RUN echo $(cat /etc/apt/sources.list)
RUN grep -v "https://apt.kitware.com/ubuntu" /etc/apt/sources.list > tmpfile &&
mv tmpfile /etc/apt/sources.list
RUN echo $(cat /etc/apt/sources.list)

RUN apt-get update && \
apt-get install -y build-essential && \
apt-get install -y libgl1 libsndfile1 && \
apt-get install -y ffmpeg && \
apt-get install -y python3-pip && \
apt-get install -y locales && \
apt-get clean

RUN sed -i '/en_US.UTF-8/s/^# //g' /etc/locale.gen && \
locale-gen
ENV LANG en_US.UTF-8
ENV LANGUAGE en_US:en
ENV LC_ALL en_US.UTF-8

RUN locale
```

```
COPY yolosed/models/yolov4.cfg /app/models/
COPY yolosed/models/yolov4.weights /app/models/
COPY yolosed/yolo/yolov4_id_to_labels.json /app/models/
COPY yolosed/maps/cutout_map.png /app/maps/
COPY yolosed/docker/requirements.txt /tmp/
RUN python3 -m pip install pillow==8.3
RUN python3 -m pip install --default-timeout=100 -r /tmp/requirements.txt
COPY yolosed/docker/mudas-0.3.9-py3-none-any.whl /tmp/
RUN python3 -m pip install --default-timeout=100 /tmp/mudas-0.3.9-py3-none-
any.whl --no-dependencies
WORKDIR /app
COPY yolosed/tau_sed/__init__.py /app/tau_sed/
COPY yolosed/tau_sed/tau_audio_ai.py /app/tau_sed/
COPY yolosed/utils.py /app/
COPY yolosed/data_receiver.py /app/
COPY yolosed/fusion_logic.py /app/
COPY yolosed/main.py /app/
COPY yolosed/yoload.py /app/
CMD ["sh", "-c", "echo PARAMETERS:; \
echo ------; \
echo ACCESS_AV_REGISTRY=${ACCESS_AV_REGISTRY};\
echo APPLY_MAP=${APPLY_MAP};\
echo AUDIO_DEVICE=${AUDIO_DEVICE};\
echo AUDIO_URL=${AUDIO_URL};\
echo BROKER=${BROKER};\
echo CAMERA_ID=${CAMERA_ID};\
echo CONFIDENCE=${CONFIDENCE};\
echo DETECTED_BY=${DETECTED_BY};\
echo EVENT_TYPE=${EVENT_TYPE};\
echo FPS=${FPS};\
echo FUSION=${FUSION};\
echo FUSION_THRESHOLD=${FUSION_THRESHOLD};\
echo GET_MODEL_REPO=${GET_MODEL_REPO};\
echo ID=${ID};\
echo LIMIT=${LIMIT};\
echo LOGGING_LEVEL=${LOGGING_LEVEL};\
echo MODE=${MODE};\
echo MODEL_NAME=${MODEL_NAME};\
echo MQTT_PASSWORD=${MQTT_PASSWORD};\
echo MQTT_PORT=${MQTT_PORT};\
echo MQTT_USERNAME=${MQTT_USERNAME};\
echo OWNER=${OWNER};\
echo QUEUE_SIZES=${QUEUE_SIZES};\
echo REGISTRY_URL=${REGISTRY_URL};\
echo SECOND_CAMERA_ID=${SECOND_CAMERA_ID};\
echo TARGET_SAMPLE_RATE=${TARGET_SAMPLE_RATE};\
echo THRESHOLD=${THRESHOLD};\
```

```
echo TOPIC=${TOPIC};\
echo USE_GPU=${USE_GPU};\
echo VERBOSE=${VERBOSE};\
echo VIDEO_URL=${VIDEO_URL};\
echo WATCH_LIST=${WATCH_LIST};\
echo WORKER_COUNT=${WORKER_COUNT};\
echo AUDIO_DEVICE=${AUDIO_DEVICE};\
echo AUDIO_CONTEXT_LENGTH=${AUDIO_CONTEXT_LENGTH};\
python3 main.py \
--access_av_registry ${ACCESS_AV_REGISTRY} \
--apply_map ${APPLY_MAP} \
--audio_device ${AUDIO_DEVICE} \
--audio_url ${AUDIO_URL} \
--broker ${BROKER} \
--camera_id ${CAMERA_ID} \
--confidence ${CONFIDENCE} \
--detected_by ${DETECTED_BY} \
--event_type ${EVENT_TYPE} \
--fps ${FPS} \
--fusion ${FUSION} \
--fusion_threshold ${FUSION_THRESHOLD} \
--get_model_repo ${GET_MODEL_REPO} \
--id ${ID} \
--limit ${LIMIT} \
--logging_level ${LOGGING_LEVEL} \
--mode ${MODE} \
--model_name ${MODEL_NAME} \
--mqtt_password ${MQTT_PASSWORD} \
--mqtt_port ${MQTT_PORT} \
--mqtt_username ${MQTT_USERNAME} \
--owner ${OWNER} \
--queue_sizes ${QUEUE_SIZES} \
--registry_url ${REGISTRY_URL} \
--second_camera_id ${SECOND_CAMERA_ID} \
--target_sample_rate ${TARGET_SAMPLE_RATE} \
--threshold ${THRESHOLD} \
--topic ${TOPIC} \
--use_gpu ${USE_GPU} \
--verbose ${VERBOSE} \
--video_url ${VIDEO_URL} \
--watch_list ${WATCH_LIST} \
--worker_count ${WORKER_COUNT} \
--audio_context_length ${AUDIO_CONTEXT_LENGTH} \
"]
```

Listing 5: YOLO-SED Dockerfile

2.1.7 CATFlow

CATFlow, developed by GRN, is a software asset that takes a video stream as input and provides a list of traffic objects tracked within the camera's field of view. It specifically classifies vehicles into six categories: cars, buses, light goods vehicles, heavy goods vehicles, bicycles, and motorcycles. Each object, whether it's a vehicle or a pedestrian, is tracked, and its trajectory is extracted and stored for visualisation or further analysis.

To protect GRN's intellectual property (IP) related to the CATFlow software asset, the CATFlow configurator was uploaded to the MARVEL registry. This configurator is responsible for retrieving the CATFlow image from GRN's Azure registry and deploying it onto the device. This ensures the secure and controlled distribution of the CATFlow software asset while safeguarding GRN's proprietary technology.

The CATFlow image consists of two main files: Dockerfile-base and Dockerfile. Dockerfile-base is responsible for building an image that includes all the required libraries to run CATFlow, such as ffmpeg, CUDNN, Python, and OpenCV. The base image is derived from NVIDIA's cuda image to minimise build time, as these libraries are not frequently updated. Once the base image is built, the CATFlow code is added to it. The code is written in Python but compiled into Cython. These steps, excluding the base image build, are automated through GRN's continuous integration/continuous deployment (CI/CD) pipeline, ensuring that the image is always up-to-date.

Similarly, the Configurator image follows a similar approach. The Configurator pulls the pre-built CATFlow image from GRN's repository using Docker commands. This image also follows the CI/CD pipeline, ensuring its freshness and alignment with the latest CATFlow version.

Both images are built on an Ubuntu 20.04 system, utilising CUDA 11.2.1, CUDNN 8, and Python 3.9. Moreover, the CATFlow image makes use of OpenCV 4.5.2. It's important to address the computing capability of the GPU used by CATFlow, as it impacts the base image. This is due to the CUDA architectures compiled by OpenCV during the image-building process. The specific Dockerfile for CATFlow is not provided in the document for privacy reasons, as it aims to avoid exposing sensitive information about the component and its input sources.

2.1.8 Text Anomaly Detection – TAD

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 (2nd Release of the MARVEL Framework) 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.

The corresponding TAD image is built on an Ubuntu 20.04 system with Python 3.9. The packages required are installed through the docker file. Listing 7 shows the commands for the manual deployment of TAD using the created Docker image.

```
docker login registry.marvel-platform.eu
docker pull registry.marvel-platform.eu/tad:0
docker run -it registry.marvel-platform.eu/tad:0 /bin/sh
```

Listing 6: TAD container creation commands

DRAFT

The first command logins the user in the MARVEL platform registry. A pull command follows that fetches the available image, while the run command creates the corresponding container from the downloaded image. As a result, a TAD container is created and started.

2.1.9 Rule Based Anomaly Detection – RBAD

The RBAD component detects anomalies, as defined by human expert-created rules. Those rules come in the form of maps drawn over the camera image (they define areas where certain objects cannot be present, for example, pedestrians on a road), bus schedule (RBAD checks whether buses arrive within the expected time) and definition of rush hours (heavyweight vehicles during rush hours are reported as anomalies). The input to this component is CATFlow messages, which contain the location, type and trajectory of detected objects. The output is a message containing the type of the anomaly detected.

The base image for the RBAD component is the Pytorch:1.12.0 container (Listing 7). The next commands set the time zone, required to properly synchronise with the other MARVEL components. The RUN command installs prerequisite software. Then, the model, python requirements are copied and installed within the image. Finally, the last command starts the component, passing the required parameters.

```
FROM pytorch/pytorch:1.12.0-cuda11.3-cudnn8-runtime

ARG DEBIAN_FRONTEND=noninteractive
ENV TZ=Europe/Copenhagen
RUN apt-get update && \
    apt-get install -y build-essential && \
    apt-get install -y libgl1 libsndfile1 && \
    apt-get install -y ffmpeg && \
    apt-get clean
COPY containers/container_source/requirements_rbad.txt /tmp/
RUN pip install --default-timeout=100 -r /tmp/requirements_rbad.txt
WORKDIR /app
COPY containers/container_source/maps/map_bicycle_not_on_lane_grn-va-01.png
/app/temp/
COPY containers/container_source/maps/map_pedestrian_jaywalking_grn-va-01.png
/app/temp/
COPY containers/container_source/maps/map_bicycle_not_on_lane_grn-va-02.png
/app/temp/
COPY containers/container_source/maps/map_pedestrian_jaywalking_grn-va-02.png
/app/temp/
COPY containers/container_source/maps/map_bicycle_not_on_lane_grn-va-03.png
/app/temp/
COPY containers/container_source/maps/map_pedestrian_jaywalking_grn-va-03.png
/app/temp/
COPY containers/container_source/maps/bus_schedule.csv /app/temp/
COPY containers/container_source/utils_rbad.py /app/
COPY containers/container_source/source_rbad.py /app/
CMD ["sh", "-c", "echo PARAMETERS:; \
    echo ------; \
    echo BROKER=${BROKER};\
```

```
    echo SOURCE_BROKER=${SOURCE_BROKER};\
    echo CAMERAID=${CAMERAID};\
    echo DETECTED_BY=${DETECTED_BY};\
    echo EVENT_TYPE=${EVENT_TYPE};\
    echo ID=${ID};\
    echo LIMIT=${LIMIT};\
    echo LOGGING_LEVEL=${LOGGING_LEVEL};\
    echo MQTT_PASSWORD=${MQTT_PASSWORD};\
    echo SOURCE_MQTT_PASSWORD=${SOURCE_MQTT_PASSWORD};\
    echo MQTT_USERNAME=${MQTT_USERNAME};\
    echo SOURCE_MQTT_USERNAME=${SOURCE_MQTT_USERNAME};\
    echo MODE=${MODE};\
    echo OWNER=${OWNER};\
    echo MQTT_PORT=${MQTT_PORT};\
    echo SOURCE_PORT=${SOURCE_PORT};\
    echo TARGET_TOPIC=${TARGET_TOPIC};\
    echo SOURCE_TOPIC_VEHICLES=${SOURCE_TOPIC_VEHICLES};\
    echo SOURCE_TOPIC_PEDESTRIANS=${SOURCE_TOPIC_PEDESTRIANS};\
    echo VERBOSE=${VERBOSE};\
    echo RUSH_HOURS=${RUSH_HOURS};\
    echo BUS_THRESHOLD=${BUS_THRESHOLD};\
    echo VIDEO_SIZE=${VIDEO_SIZE};\
    python source_rbad.py \
        --broker ${BROKER} \
        --source_broker ${SOURCE_BROKER} \
        --cameraid ${CAMERAID} \
        --detected_by ${DETECTED_BY} \
        --event_type ${EVENT_TYPE} \
        --id ${ID} \
        --limit ${LIMIT} \
        --logging_level ${LOGGING_LEVEL} \
        --mqtt_password ${MQTT_PASSWORD} \
        --source_mqtt_password ${SOURCE_MQTT_PASSWORD} \
        --mqtt_username ${MQTT_USERNAME} \
        --source_mqtt_username ${SOURCE_MQTT_USERNAME} \
        --mode ${MODE}  \
        --owner ${OWNER} \
        --mqtt_port ${MQTT_PORT} \
        --source_port ${SOURCE_PORT} \
        --target_topic ${TARGET_TOPIC} \
        --source_topic_vehicles ${SOURCE_TOPIC_VEHICLES} \
        --source_topic_pedestrians ${SOURCE_TOPIC_PEDESTRIANS} \
        --verbose ${VERBOSE} \
        --rush_hours ${RUSH_HOURS} \
        --bus_threshold ${BUS_THRESHOLD} \
        --video_size ${VIDEO_SIZE}"]
```

Listing 7: RBAD Dockerfile

2.2 Security, privacy and data protection subsystem

2.2.1 EdgeSec Virtual Private Network – VPN

EdgeSec VPN is based on the n2n architecture, which consists of two main components: edge nodes and Super Nodes. The edge nodes utilise the Super Nodes to discover other edge nodes within the network. In addition, the Super Nodes play a crucial role in routing traffic when the nodes are located behind symmetrical firewalls. The n2n architecture operates as a peer-to-peer VPN functioning at the second layer of the OSI model. This allows the peers to establish connectivity across network address translation (NAT) devices and firewalls. The edge nodes that belong to the same virtual network form a community, while Super Nodes can serve multiple communities. Furthermore, a single computer can join multiple communities in the network. The functionality of EdgeSec VPN is described in detail in D4.2[2].

EdgeSec VPN is unchanged since its first version. One important aspect to note is that for each new node added to the MARVEL Kubernetes cluster, EdgeSec VPN needs to be deployed. This means that whenever a new node is introduced to the cluster, the EdgeSec VPN solution must be installed and configured on that specific node. This ensures that the newly added node can participate in the secure and private network established by EdgeSec VPN within the MARVEL Kubernetes environment. By deploying EdgeSec VPN to each new node, the network connectivity and communication between the nodes can be established in a protected manner, maintaining the overall security and integrity of the cluster.

2.2.2 EdgeSec Trusted Execution Environment – TEE

The primary goal of EdgeSec Trusted Execution Environment (TEE) is to offer a secure and confidential execution environment for sensitive data processing applications and components within MARVEL. It achieves this by utilising Intel Software Guard Extensions (SGX) technology, which is specifically supported by a subset of Intel processors. To enhance security even within containers, EdgeSec TEE integrates with Secure Container Environment (SCONE), a software platform specifically designed to safeguard the data and code of applications running in Linux containers. This integration ensures that the execution of sensitive applications within containers remains protected and confidential.

With Intel SGX, developers can partition their applications into two parts: a sensitive portion that requires integrity and data protection, and a non-sensitive portion. The trusted execution provided by Intel SGX involves isolating the sensitive code within encrypted memory regions known as memory enclaves. Furthermore, Intel SGX offers remote attestation, a security feature that ensures the enclave's integrity before transmitting data to it.

In R2, the combination of EdgeSec TEE and VideoAnony becomes relevant due to the handling of sensitive data like camera feed URLs, usernames, and passwords. A simple Hypertext Transfer Protocol (HTTP) service based on Python Flask was created, where the endpoints of this service provide sensitive information to VideoAnony. The concept of EdgeSec TEE is applied to the mentioned HTTP service. Through the process of sconification, the container running the HTTP service is executed within protected private memory regions known as enclaves.

---

[2] MARVEL D4.2 - Security assurance and acceleration in E2F2C framework – initial version, 2022. https://doi.org/10.5281/zenodo.6821254

DRAFT

For the deployment of EdgeSec TEE a new template was created where 3 containers are instantiated in a single pod. The first container is the sconified image where the sensitive data is encrypted, the second one is part of the attestation process and the last one is for the VideoAnony. For further details concerning the sconification process as well as the attestation please refer to D4.5[3].

```
# edgesecTEE.template.yaml
apiVersion: v1
kind: Service
metadata:
  name: $NAME
spec:
  type: ClusterIP
  ports:
  - port: 8554
    targetPort: rtsp
    protocol: TCP
    name: rtsp
  selector:
    app: $NAME
---
apiVersion: apps/v1
kind: Deployment
metadata:
  creationTimestamp: null
  labels:
    app: $NAME
  name: $NAME
spec:
  replicas: 1
  selector:
    matchLabels:
      app: $NAME
  strategy: {}
  template:
    metadata:
      creationTimestamp: null
      labels:
        io.kompose.network/sconify-image-default: "true"
        app: $NAME
    spec:
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
```

[3] MARVEL D4.5 - Security assurance and acceleration in E2F2C framework – final version, 2023. To appear.

```yaml
            nodeSelectorTerms:
            - matchExpressions:
              - key: Layer
                operator: In
                values:
                - UNSEDGE1
      containers:
        - image: registry.scontain.com:5050/sconecuratedimages/kubernetes:las-
scone5.1
          name: las
          ports:
            - containerPort: 18876
          resources: {}
          securityContext:
            privileged: true
        - args:
            - sh
            - -c
            - sleep 15; export SCONE_LOG=7 ; export SCONE_LAS_ADDR=localhost ;
export SCONE_CAS_ADDR=scone-cas.cf; export SCONE_CONFIG_ID=my_namespace-
8617/flask/service ; echo SESSION=my_namespace-8617/flask SCONE_HEAP=1G ; export
SCONE_STACK=4M ;export SCONE_HEAP=1G ; export SCONE_ALLOW_DLOPEN=2 ;
/usr/local/bin/python
          image: 192.168.50.1:5000/flask_restapi_image:0.7
          name: python
          ports:
            - containerPort: 4996
          resources: {}
          securityContext:
            privileged: true
        - name: $NAME
          image: 192.168.50.1:5000/videoanonycputee:0.4
          command: [sh, -c]
          args: ["until python3 src/anonymize.py --source CAM_UNS_CCTV_01 --tee
True --head-model /private/weights/crowdhuman1280x_yolov5s.pt --lpd-model
/private/weights/grn1280x_yolov5s.pt --vstream-uri rtsp://rtspserver-grnedge1-
nodeport.karvdash-lucadvlfbk.svc:8554/GRN-VA-01_Mgarr_VideoAnony --stream-fps 2;
do echo 'Restarting ffmpeg command...'; sleep 10; done"]
      tolerations:
       - key: "Layer"
         operator: "Equal"
         value: "UNSEDGE1"
      restartPolicy: Always
status: {}
---
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
```

```
metadata:
  creationTimestamp: null
  name: sconify-image-default
spec:
  ingress:
    - from:
        - podSelector:
            matchLabels:
              io.kompose.network/sconify-image-default: "true"
  podSelector:
    matchLabels:
      io.kompose.network/sconify-image-default: "true"
---
kind: Template
name: Deploy EdgeSec TEE on UNS1 with VideoAnony
description: Deploy EdgeSec TEE on UNS1
variables:
- name: NAME
  default: edgesectee-uns1
- name: PRIVATE_VOLUME
  default: private-volume
```

Listing 8: TEE & VideoAnony combined template YAML file

2.2.3 VideoAnony

The goal of the component is to read the raw video streams from the cameras and anonymise faces and car plates via blurring. The component has not been changed since D3.2. Note that in the MT use cases, the component is deployed outside MARVdash for privacy restrictions introduced by MT's DPO.

2.2.4 AudioAnony & VAD (devAIce)

This component represents the MARVEL audio anonymisation pipeline. Its goal is to ingest and process, in real-time, audio streams to detect sensitive audio data, mainly manifesting in speech segments. This is done via the voice activity module. These segments are further processed to be anonymised with signal-processing-based techniques; this is done with AudioAnony. The processed segments, anonymised or not, are forwarded to the later stages of the MARVEL audio-visual pipeline, to be consumed and used by the pool of the audio AI components available. The component also keeps track of the analysis by forwarding the event's boundaries to the related Message Queuing Telemetry Transport (MQTT) broker, which will be later stored in the Elastic Search database and consumed by SmartViz.

The component is deployed as a docker image via MARVdash in UNS use cases, while in MT use cases it is installed outside the Kubernetes cluster.

The current version of the component has not been changed since D3.2. Changes have been applied to read the new 8-channel microphone arrays but without modifying the functionality and deployment process.

2.3 Data management and distribution subsystem

2.3.1 StreamHandler

INTRA’s StreamHandler Platform provides all the necessary functionalities of a modern scalable big data platform. The core functionality of StreamHandler has been completely reconstructed compared to the StreamHandler version presented in D3.2.

In the context of the MARVEL framework, the StreamHandler’s core functionality deployed in the R2 phase consists of four services:

- 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 REpresentational State Transfer (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 - Store binary files, it is used for storing, retrieving and sharing the produced audio-visual files in binary format.

Below, in Listing 9 and Listing 10, the corresponding Ain't Markup Language (YAML) files from the GRN Fog 2 use case deployment is depicted.

```
# media-content-service-grn.yaml
---
apiVersion: v1
kind: Service
metadata:
  name: $NAME
spec:
  type: ClusterIP
  selector:
    app: $NAME
  ports:
    - name: http
      protocol: TCP
      port: 8889
      targetPort: 8080
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: $NAME
spec:
  replicas: 1
  selector:
    matchLabels:
      app: $NAME
  template:
    metadata:
      labels:
```

```
        app: $NAME
    spec:
      containers:
      - name: $NAME
        image: 192.168.50.1:5000/media-content-service:1.0
        volumeMounts:

            - name: ${PRIVATE_VOLUME}
              mountPath: /conf.json
              subPath: conf/conf.json
        ports:
        - containerPort: 8080
      affinity:
        nodeAffinity:
         requiredDuringSchedulingIgnoredDuringExecution:
          nodeSelectorTerms:
          - matchExpressions:
            - key: Layer
              operator: In
              values:
              - GRNFOG2
      tolerations:
      - key: "Layer"
        operator: "Equal"
        value: "GRNFOG2"
---
kind: Template
name: media-content-services-grn
description: media-content-service-grn
variables:
- name: NAME
  default: media-content-service-grn
- name: PRIVATE_VOLUME
  default: private-volume
```

Listing 9: Service 2 Docker file for GRN Fog 2 deployment

```
# rtsp-stream-handler-grn.yaml
---
apiVersion: v1
kind: Service
metadata:
  name: $NAME
spec:
  type: ClusterIP
  selector:
    app: $NAME
  ports:
```

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

            - name: ${PRIVATE_VOLUME}
              mountPath: /conf.json
              subPath: conf/conf.json
        ports:
        - containerPort: 8080
      affinity:
        nodeAffinity:
         requiredDuringSchedulingIgnoredDuringExecution:
          nodeSelectorTerms:
          - matchExpressions:
            - key: Layer
              operator: In
              values:
              - GRNFOG2
      tolerations:
      - key: "Layer"
        operator: "Equal"
        value: "GRNFOG2"
---
kind: Template
name: rtsp-stream-handler-grn
description: rtsp-stream-handler-grn
variables:
- name: NAME
```

```
  default: rtsp-stream-handler-grn
- name: PRIVATE_VOLUME
  default: private-volume
```

Listing 10: Service 1 Docker file for GRN Fog 2 deployment

For more information about StreamHandler, the reader is referred to deliverable D2.4[4].

2.3.2 Data Fusion Bus – DFB

The DFB component has not undergone any significant changes after the time of D3.2 preparation and therefore the information in Section 2.3.2 of D3.2 is still applicable. The main changes are described below.

The following services were added to the DFB deployment as running pods within the MARVEL Kubernetes cluster:

- **Fusion service**. This service is responsible for consuming messages from the Kafka cluster that originate from the AVAD, ViAD, SED, and AT components, applying fusion and elimination rules on these messages and re-publishing the resulting messages on Kafka and permanently storing them on the Elastic Search repository.
- **Kafdrop**. A simple Web User Interface (UI) tool for monitoring the messages published on the Kafka cluster was deployed. The Kafdrop tool is developed by Obsidian Dynamics and is available at the following Github repo[5].

The Kafka service has been modified to expose the Kafka cluster to other services that reside outside of the MARVEL Kubernetes cluster using the NodePort method. The relevant changes in the YAML file used for the Kafka service deployment configuration are presented in Listing 11 below.

```
apiVersion: v1
kind: Service
metadata:
  name: kafka-broker-0
spec:
  type: NodePort
  ports:
  - port: 30090
    targetPort: 30090
    nodePort: 30090
    protocol: TCP
  selector:
    statefulset.kubernetes.io/pod-name: cp-kafka-0
---
apiVersion: v1
kind: Service
metadata:
  name: kafka-broker-1
spec:
  type: NodePort
```

[4] MARVEL D2.4 - Management and distribution Toolkit – final version, 2023. To appear.

[5] https://github.com/obsidiandynamics/kafdrop

```yaml
  ports:
  - port: 30091
    targetPort: 30091
    nodePort: 30091
    protocol: TCP
  selector:
    statefulset.kubernetes.io/pod-name: cp-kafka-1
---
apiVersion: v1
kind: Service
metadata:
  name: kafka-broker-2
spec:
  type: NodePort
  ports:
  - port: 30092
    targetPort: 30092
    nodePort: 30092
    protocol: TCP
  selector:
    statefulset.kubernetes.io/pod-name: cp-kafka-2
```

Listing 11: Use of NodePort method in the deployment YAML configuration file for the DFB Kafka service.

Furthermore, the DataFusion connector service has been modified to include the Kibana tool that is connected to the deployed Elastic Search repository. In addition, the Kafka topics to be monitored for transferring the published messages to the permanent Elastic Search storage have been exposed as parameters in the YAML file used for the configuration of the service (relevant excerpt in Listing 12 below).

```yaml
            - name: topicsfilepath
              value: "/root/es-connector/topics.properties"
            - name: topicstoupdate
              value: "CATFlow-V,CATFlow-P, TAD, ViAD, AVAD, VCC, AVCC, SED, AT, 
VAD, AAC, RBAD, YOLO-SED, SELD, GPURegex"
            - name: elasticsearch.host
              value: "es-es-http"
            - name: elasticsearch.port
              value: "9200"
            volumeMounts:
              - name: ${PRIVATE_VOLUME}
                mountPath: /root/es-connector
                subPath: .es-connector
```

Listing 12: Exposing the Kafka topics to be monitored by the DataFusion connector service as parameters.

2.3.3 DatAna

DatAna is a Data Management Platform acting as a bridge to collect the results of the inference in the different layers of the pilots (edge, fog, and cloud infrastructure). The main usage of DatAna in the E2F2C MARVEL infrastructure was explained in D3.2 and has not changed substantially for the R2 (Release #2 of the MARVEL Framework). A summary of the usage of DatAna along with some minimal updates is explained below.

DatAna is an Apache NiFi-based tool working in combination with MQTT brokers in the different layers to collect, validate and transform the inference data to the agreed data models

for Alert, Anomaly, and MediaEvent, inherited from the Smart Data Models initiative [1] and extended for MARVEL purposes. The results of these transformations are sent to the cloud and populated in the Kafka instance of the DFB for further fusion and storage.

DatAna is deployed as docker containers in the different layers where it is needed to manage results from inference using MARVdash. NiFi docker containers and YAML files have been provided and fine-tuned, as well as the secure mechanisms implemented by NiFi to connect the different instances via the secured Site-to-Site (S2S) protocol. The use of TLS certificates has been enabled among the instances, on a client-server basis to enable the S2S secure communication among the network of NiFi instances.

Figure 1 shows graphically the network or topology of deployment of NiFi instances and MQTT brokers in the different layers for the pilots' use cases in R2. It is worth noticing that in some cases at the edge, a decision was made to deploy only MQTT and collect the results from the instance of NiFi at the fog, mainly to not overload edge devices with more components deployed in them. Therefore, the way of deploying the NiFi services using MARVdash has not changed in the second half of the project, but only the infrastructure where DatAna is deployed.

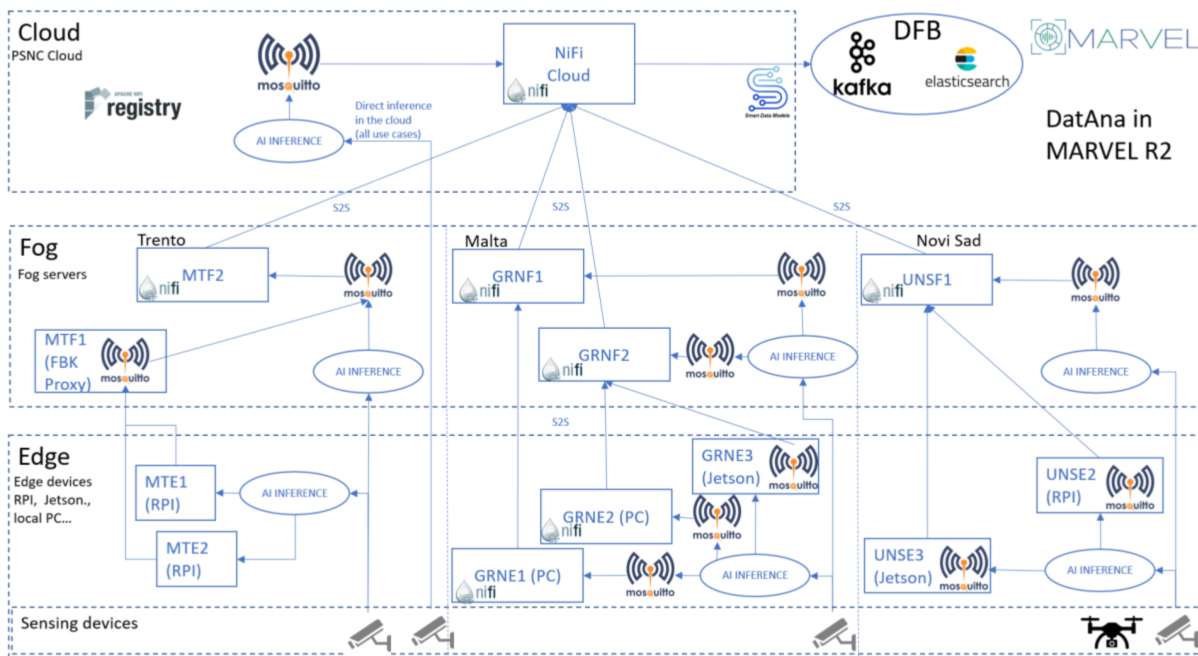

Figure 1. DatAna topologies for R2

2.3.4 Hierarchical Data Distribution – HDD

HDD is an algorithmic framework to optimise the configuration of an Apache Kafka cluster, in particular, by determining for a given topic the number of brokers and partitions that match the hard constraints of the application on the replication latency and unavailability time, while using the system resources efficiently, e.g., in terms of the OS load. In principle, HDD can be used in a closed-loop manner within an edge-cloud domain as follows: 1) a background service waits for the current configuration of the applications and Apache Kafka topics; 2) when there are any changes, the background service is triggered and provided with the new data; 3) it runs HDD, which currently supports two approximation algorithms, called BroMin and BroMax, which have been evaluated through simulation [2]; 4) it returns the new configuration parameters to the caller, which can take appropriate measures to optimise the run-time configuration of the Apache Kafka cluster. We have showcased this approach by deploying the

Docker image registry.marvel-platform.eu/hddv0:2, which is available as a service in MARVdash, as already described in D3.2.

2.4 E2F2C subsystem

2.4.1 GPURegex

GPURegex can be deployed to any OpenCL-enabled processor or hardware accelerator (such as a discrete GPU or a shared GPU). As already mentioned in D3.2, OpenCL drivers are required to be installed in the specific docker container before the execution of GPURegex. Each vendor (e.g., Intel, NVIDIA) and each hardware device (e.g., CPU, discrete GPU, integrated GPU) is supported by the vendor and device-specific OpenCL drivers, libraries and runtimes. In the first version of this deliverable (D3.2), GPURegex was offered via two different docker images. The first image was tailored for Intel CPUs only (for hardware setups that do not include a GPU), while in the second image, GPURegex could be executed on top of either the CPU or the shared GPU (Intel HD Graphics). After M18, GPURegex for NVIDIA GPUs has been uploaded to the MARVEL docker image registry. This container can be used for hardware setups that include an Intel CPU and an NVIDIA GPU.

Once downloaded from the MARVEL docker image registry, the GPURegex component can be deployed following the steps presented in Listing 13.

```
docker login registry.marvel-platform.eu

docker pull registry.marvel-platform.eu/gpuregex-intel-gpu:4

docker run -it registry.marvel-platform.eu/gpuregex-intel-gpu:4 /bin/sh
```

Listing 13: GPURegex container creation commands

GPURegex is compiled and executed using the commands presented in Listing 14. Option "`-d 1`" indicates the device position that we wish to use for execution; in this case, the position of the NVIDIA GPU is 1. The devices and their positions can be found via the command "`clinfo -l`". With the option "`-m 0`", we instruct the program that the device selected for execution does not share the same memory address space with the CPU. In the case of an integrated GPU or CPU, we must use the option "`-m 1`", which is the default selection if not specified. Options "`-p`" and "`-i`" accept the pattern file and input file, respectively. GPURegex reports any input entries that match at least one of the patterns contained in the pattern file.

```
$ make

$ ./bin/gpuregex -d 1 -m 0 -p patterns.dat -i TheAdventuresOfSherlockHolmes.dat
```

Listing 14: GPURegex execution commands

Finally, the template that was used for the deployment configuration of the newly added image is displayed in Listing 15.

```
# gpuregex-priviledged-MTFOG2.template.yaml
apiVersion: v1
kind: Service
metadata:
  name: $NAME
spec:
```


DRAFT

```
  type: ClusterIP
  ports:
  - port: 8080
  selector:
    app: $NAME
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: $NAME
spec:
  replicas: 1
  selector:
    matchLabels:
      app: $NAME
  template:
    metadata:
      labels:
        app: $NAME
    spec:
      affinity:
        nodeAffinity:
         requiredDuringSchedulingIgnoredDuringExecution:
          nodeSelectorTerms:
          - matchExpressions:
            - key: Layer
              operator: In
              values:
              - MTFOG2
      containers:
      - name: $NAME
        image: 192.168.50.1:5000/gpuregex-intel-gpu:4
        command: [ "sleep" ]
        args: [ "infinity" ]
        securityContext:
          privileged: true
        ports:
        - containerPort: 8080
        env:
        - name: MESSAGE
          value: $MESSAGE
        resources:
          limits:
            aliyun.com/gpu-mem: 1
      tolerations:
      - key: "Layer"
        operator: "Equal"
        value: "MTFOG2"
```

```
---
kind: Template
name: GPURegex MTFOG2 Intel/NVIDIA priviledged
description: GPURegex running in MTFOG2 (Intel Xeon CPU & NVIDIA GPU)
priviledged
variables:
- name: NAME
  default: gpuregex-priviledged2-mtfog2
- name: HOSTNAME
  default: gpuregex-priviledged2-mtfog2.example.com
```

Listing 15: Template used to deploy the newly added GPURegex image (Intel CPU and NVIDIA GPU)

2.4.2 DynHP

DynHP is a methodology to compress Deep Neural Networks (DNN) via pruning. This methodology has been developed having in mind the edge scenario where an edge device needs to compress a model using its data. To this end, it is designed as an iterative process where a DNN is trained and compressed incrementally implementing the Hard-Pruning paradigm, i.e., when a group of parameters is removed it cannot be recovered in the future. This feature allows, at least in principle, to run it and, at the same time, free memory resources from the edge device.

Concerning D3.2, the algorithmic implant has not changed. The main modifications regarded the application of this approach to other models, e.g., AVCC. This required complete code refactoring to make it easily extensible and usable by third parties. Since DynHP is not a service but a library, it has been publicly released on GitHub.

2.4.3 FedL

FedL has been delivered in the previous version of the MARVEL framework. The details of the component, container images and deployment have been provided in D3.3. The component has been applied in use cases MT1 (Monitoring of Crowded Areas), GRN4 (Junction Traffic Trajectory Collection), and UNS1 (Drone Experiment), with visual crowd-counting model incorporated. For R2 use cases, FedL has also been further developed for the audio-visual emotion recognition (AVER) model within (the former) UNS2 use case AVER[6]. The details of this development are provided in D3.5[7].

2.5 System outputs subsystem

2.5.1 SmartViz

The system outputs of MARVEL are realised through the Decision-Making Toolkit (DMT) which aims at assisting the stakeholders in short and long-term decision-making. The DMT is based on the SmartViz component created by ZELUS, which contains a collection of visualisation widgets, offering multi-purpose advanced data representations and visualisations.

[6] We remark that the former UNS2 use case AVER was meanwhile replaced by a new UNS2 use case focused on Sound event localization and detection in crowds. The rationale of this change is provided in D6.3.

[7] MARVEL D3.5 - Multimodal and privacy-aware audio- visual intelligence – final version, 2023. To appear.

DRAFT

SmartViz is a versatile data visualisation toolkit that allows domain experts and simple users to discover behaviours and correlations of data items. The toolkit visualises both real-time and historical data, and it is configured according to the MARVEL stakeholders' needs.

Using its Data Intake adapters, SmartViz can connect with multiple data sources and then uses its internal data API and configuration options to produce predefined as well as user-defined visualisation dashboards. The output of the adapters is handled by a middleware, that transforms information into internal data representations, which can afterwards feed the visualisations.

In the internal architecture of SmartViz in R2 there is an addition of a connection with MARVdash, which introduces a new functionality that empowers users with service-control capabilities. This functionality is achieved through the interaction of SmartViz and MARVdash via a REST API. Through this REST API, SmartViz can communicate with MARVdash and retrieve information about the services that are currently deployed within MARVEL. Based on this information, SmartViz allows users to initiate or terminate selected services within the system.

The Frontend part of the tool is served as a web application directly accessible by end-users. SmartViz is under a single deployment in the cloud, and it is utilised in all the use cases. Since the previous version of this deliverable (D3.2), this toolkit has no major changes regarding its deployment, besides the support of new connections with some of MARVEL's components and its services.

SmartViz consists of two images, one for the Frontend and one for the Middleware part, and it also uses an NGINX web server to act as a proxy for the internal services. There are two Dockerfiles as seen below (Listing 16) for each part of the application and a docker-compose.yaml file.

```
Middleware Dockerfile:
FROM node:14
WORKDIR /usr/src/app/srv
COPY package.json package-lock.json ./
RUN npm install
COPY . .
CMD [ "node", "server.js" ]

Frontend Dockerfile:
FROM node:14-alpine As builder
WORKDIR /usr/src/app/smartviz
COPY package.json package-lock.json ./
RUN npm install
COPY . .
RUN npm run build -- --base-href='/smartviz/'
FROM nginx:alpine
COPY --from=builder /usr/src/app/smartviz/dist /usr/share/nginx/html/smartviz

CMD ["/bin/sh", "-c", "envsubst <
/usr/share/nginx/html/smartviz/assets/env.template.js >
/usr/share/nginx/html/smartviz/assets/env.js && exec nginx -g 'daemon off;'"]

Docker-compose yaml:
```

```yaml
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  annotations:
    nginx.ingress.kubernetes.io/force-ssl-redirect: '"true"'
    nginx.ingress.kubernetes.io/proxy-read-timeout: "3600"
    nginx.ingress.kubernetes.io/proxy-send-timeout: "3600"
    nginx.ingress.kubernetes.io/proxy-body-size: "0"
  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
  selector:
    app: $NAME
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: nginx-config
data:
  default.conf: |
    upstream server {
        server 127.0.0.1:8000;
    }
    upstream smartviz {
        server 127.0.0.1:8080;
    }
    upstream serverws {
        server 127.0.0.1:31016;
    }

    server {
        listen 80;
        location /smartviz {
```

```
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_set_header X-Forwarded-Proto $scheme;
            proxy_http_version 1.1;
            proxy_set_header Upgrade $http_upgrade;
            proxy_set_header Connection "upgrade";
            proxy_pass http://smartviz;
        }
        location /server {
            rewrite ^/server/(.*) /$1 break;
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_set_header X-Forwarded-Proto $scheme;
            proxy_http_version 1.1;
            proxy_set_header Upgrade $http_upgrade;
            proxy_set_header Connection "upgrade";
            proxy_pass http://server/;
        }
        location /apiProxy {
            rewrite ^/apiProxy/(.*) /$1 break;
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_set_header X-Forwarded-Proto $scheme;
            proxy_pass http://datafusion-es-proxy.karvdash-ikyrannas.svc:8080;
        }
        location /apiMarvSer {
            rewrite ^/apiMarvSer/(.*) /$1 break;
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_set_header X-Forwarded-Proto $scheme;
            proxy_pass http://karvdash.default.svc/api/services/;
        }

        location /AVProxyMT {
            rewrite ^/AVProxyMT/(.*) /$1 break;
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_set_header X-Forwarded-Proto $scheme;
            proxy_pass http://avregistry-mt.karvdash-tkanellos:3000;
        }
        location /AVProxyGRN {
            rewrite ^/AVProxyGRN/(.*) /$1 break;
            proxy_set_header Host $host;
```

```
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_set_header X-Forwarded-Proto $scheme;
            proxy_pass http://avregistry-grn.karvdash-tkanellos:3000;
        }
        location /AVProxyUNS {
            rewrite ^/AVProxyUNS/(.*) /$1 break;
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_set_header X-Forwarded-Proto $scheme;
            proxy_pass http://avregistry-uns.karvdash-tkanellos:3000;
        }
        location /StreamHandlerGRN {
            rewrite ^/StreamHandlerGRN/(.*) /$1 break;
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_set_header X-Forwarded-Proto $scheme;
            proxy_pass http://media-content-service-grn.karvdash-manf.svc:8889;
        }
        location /MinioGRN {
            rewrite ^/MinioGRN/(.*) /$1 break;
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_set_header X-Forwarded-Proto $scheme;
            proxy_pass http://streamhandler-minio-server-grn-red.karvdash-
manf.svc:9000;
        }
        location /StreamHandlerMT {
            rewrite ^/StreamHandlerMT/(.*) /$1 break;
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_set_header X-Forwarded-Proto $scheme;
            proxy_pass http://media-content-service-mt.karvdash-manf.svc:8889;
        }
        location /MinioMT {
            rewrite ^/MinioMT/(.*) /$1 break;
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_set_header X-Forwarded-Proto $scheme;
            proxy_pass http://streamhandler-minio-server-mt-red.karvdash-
manf.svc:9000;
        }
        location /StreamHandlerUNS {
```

```
            rewrite ^/StreamHandlerUNS/(.*) /$1 break
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_set_header X-Forwarded-Proto $scheme;
            proxy_pass http://media-content-service-uns.karvdash-manf.svc:8889;
        }
        location /MinioUNS {
            rewrite ^/MinioUNS/(.*) /$1 break;
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_set_header X-Forwarded-Proto $scheme;
            proxy_pass http://streamhandler-minio-server-uns-red.karvdash-
manf.svc:9000;
        }
        location /serverws {


            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_set_header X-Forwarded-Proto $scheme;
            proxy_http_version 1.1;
            proxy_set_header Upgrade $http_upgrade;
            proxy_set_header Connection "upgrade";
            proxy_pass http://serverws/;
        }
      }
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: smartviz-config
data:
  default.conf: |
    server {
      listen 8080;
      location / {
        root /usr/share/nginx/html; #nginx root html
        index index.html index.htm;
        try_files $uri $uri/ /smartviz/index.html =404; #subfolder's index path
      }
      include /etc/nginx/extra-conf.d/*.conf;
    }
---
apiVersion: apps/v1
kind: Deployment
```

```
metadata:
  name: $NAME
spec:
  replicas: 1
  selector:
    matchLabels:
      app: $NAME
  template:
    metadata:
      labels:
        app: $NAME
    spec:
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
            - matchExpressions:
              - key: workerName
                operator: In
                values:
                - masterCloud
      containers:
      - name: nginx
        image: nginx:1.19.6-alpine
        ports:
        - containerPort: 80
        volumeMounts:
        - name: nginx-config-volume
          mountPath: /etc/nginx/conf.d/default.conf
          subPath: default.conf
      - name: server
        image: stellamarkop/dmtserv:$VERSIONSE
        ports:
        - containerPort: 8000
        - containerPort: 31016
        env:
        - name: PORT
          value: "8000"
        - name: KAFKA_URL
          value: $KAFKA #setting the kafka IP + Port (broker)
        - name: TOPIC
          value: $TOPIC #setting the topics
        - name: EL
          value: http://$ELASTICSEARCH #setting the elastic search IP + Port
      - name: smartviz
        image: stellamarkop/dmtsmartviz:$VERSIONSM
        ports:
        - containerPort: 8080
```

```
        env:
        - name: SERV_HOST
          value: https://$HOSTNAME/server
        - name: SOCKET_HOST
          value: https://$HOSTNAME #server base IP for Socket.io. Server runs
under nginx proxy pass. The frontend is configured to request the Socket under
/server subdomain.
        - name: ES_SVC
          value: https://$HOSTNAME/apiProxy
        - name: AV_MT_SVC
          value: https://$HOSTNAME/AVProxyMT
        - name: AV_GRN_SVC
          value: https://$HOSTNAME/AVProxyGRN
        - name: AV_UNS_SVC
          value: https://$HOSTNAME/AVProxyUNS
        - name: SH_MT_SVC
          value: https://$HOSTNAME/StreamHandlerMT
        - name: MINIO_MT_SVC
          value: https://$HOSTNAME/MinioMT
        - name: SH_GRN_SVC
          value: https://$HOSTNAME/StreamHandlerGRN
        - name: MINIO_GRN_SVC
          value: https://$HOSTNAME/MinioGRN
        - name: SH_UNS_SVC
          value: https://$HOSTNAME/StreamHandlerUNS
        - name: MINIO_UNS_SVC
          value: https://$HOSTNAME/MinioUNS
        - name: WS
          value: wss://$HOSTNAME/serverws
        - name: MARV_SVC
          value: https://$HOSTNAME/apiMarvSer
        volumeMounts:
        - name: smartviz-config-volume
          mountPath: /etc/nginx/conf.d/default.conf
          subPath: default.conf
      volumes:
      - name: nginx-config-volume
        configMap:
          name: nginx-config
          defaultMode: 0644
      - name: smartviz-config-volume
        configMap:
          name: smartviz-config
          defaultMode: 0644
---
kind: Template
name: SmartViz (DMT)
description: SmartViz
```

```
singleton: yes
datasets: no
variables:
- name: NAME
  default: smartvizR2
- name: HOSTNAME
  default: smartviz.example.com
- name: VERSIONSM
  default: smartviz_R2
  help: Container SmartViz version/tag
- name: VERSIONSE
  default: smartviz_server_R2
  help: Container Server version/tag
- name: KAFKA
  default: kafka.karvdash-ikyrannas.svc:9092
  help: Kafka service endpoint
```

Listing 16: SmartViz Dockerfiles and Docker-compose yaml

2.5.2 MARVEL Data Corpus-as-a-Service

The MARVEL Data Corpus will provide a comprehensive collection of datasets that have been anonymised and labelled. It will gain data from testing areas such as video/audio recordings from surveillance cameras, and then store it in a Big Data storehouse. This will allow users to obtain the necessary material for a more efficient utilisation of machine learning approaches. The data is available to both internal and external users, including research and commercial entities.

The core file repository of the Data Corpus-as-a-Service is still the Hadoop Distributed Files System (HDFS) but it is now installed natively on dedicated machines. Management of this Big Data database is performed through HBase which operates as a separate container.

The augmentation techniques deployed in the Corpus have been placed in a separate individual container to enable the reuse across the several nodes of the Corpus. The main Data Corpus VM is located in the MARVEL backend/cloud server and the MARVdash. It includes the elements of the Master HBase/Hadoop Node, the Name Node, the graphical interfaces, the Python augmentation libraries, and the JAVA applications that implement the programmable interfaces and the integration with other MARVEL components (i.e., DFB and StreamHandler). In addition, a separate container that handles the view-only part of the Data Corpus has been also deployed.

To sum up, the following docker images are deployed:

- HBase
    - hbase-master:1.0.0-hbase1.2.6
    - hbase-regionserver:1.0.0-hbase1.2.6
- Zookeeper
    - zookeeper:3.4.10
- Ambari
    - docker-ambari
- ELK
    - elasticsearch:elastdocker-7-17.0

DRAFT

    - logstash:elastdocker-7-17.0
    - kibana:elastdocker-7.17.0
- Python augmentations
    - augmentation_container
- JAVA application
    - docker-hbase_fileservice
- GUI
    - angular-gui_service
    - angular-gui_service for view only

DRAFT

3 E2F2C optimised deployment solution

This section is dedicated to the description of the proposed deployment solution that was developed during the lifetime of the project. In the first subsection, we describe the architecture of the MARVEL E2F2C framework. The optimisation regarding the deployment procedure is analysed in the second subsection.

3.1 Architecture of the MARVEL E2F2C framework

The MARVEL E2F2C framework is a cluster of nodes that has been created as a testbed for developing and implementing the deployment logic proposed for the AI MARVEL components. It should be mentioned that the same logic was used for the deployment of the whole set of MARVEL components. MARVEL E2F2C framework consists of nodes for all pilots and use cases.

Figure 2. Topology of MARVEL E2F2C Kubernetes cluster

These nodes are spread across three layers: edge, fog, and cloud (Figure 2). Placing some of the MARVEL framework nodes at the edge and fog layers brings several benefits such as:

- Reduction of latency in processing tasks. By bringing computational resources closer to the data producers (sensors), the transportation time is reduced and application response time becomes faster.
- Improvement of data privacy. Sensitive data are kept close to their generation environment and it is not transferred to centralised cloud hosts with questionable security.
- Real-time decision-making. Decision-making becomes faster when it takes place close to the source of data generation, making immediate responses to events possible and time-sensitive applications feasible.

- Scalability. Edge or fog nodes can scale horizontally when the user base of an application increases, ensuring high availability.

The whole set of Kubernetes nodes can be seen in Table 2 below. The offerings of edge and fog layers make them valuable additions to the cloud layer, allowing efficient and intelligent data processing. By putting together, the benefits of the edge/fog layers and those of the cloud layer (scalability, cost-effectiveness, geographic distribution, management and monitoring, etc.), we end up with a comprehensive framework where the proposed deployment logic can be applied. This deployment logic exploits the full potential of the personalised Federated Learning approach in an optimised way, taking under consideration resource availability and DL inference requirements, and enabling real-time decision-making.

Table 2: MARVEL E2F2C Kubernetes nodes

E2F2C Layer	Nodes	Description
Cloud	Master	VM
	worker1	VM
	worker2	VM
	worker3	VM
	worker4	VM
	GPU worker	Physical Machine Server with GPU
Fog	GRN fog 2	Physical Machine Server with GPU
	UNS fog 1	Physical Machine Server with GPU
	MT fog 2	Physical Machine Server with GPU
Edge	GRN edge 1	Physical Machine Desktop
	GRN edge 2	Physical Machine Desktop
	GRN edge 3	Jetson
	GRN edge 4	Jetson
	UNS edge 1	Intel NUC
	UNS edge 2	Raspberry Pi
	UNS edge 3	Physical Machine Laptop

MARVEL E2F2C framework primarily consists of the Kubernetes open-source container management platform [3] and the MARVdash service management software that acts as a Kubernetes dashboard.

Kubernetes

Kubernetes is a portable, extensible, and open-source platform designed for managing containerised workloads and services. It offers a range of benefits and features that make it a popular choice for orchestrating container-based applications.

One of the key advantages of Kubernetes is its ability to manage containers effectively. Containers are lightweight, portable, and immutable, providing a consistent environment for running applications. Kubernetes takes care of container management, including network setup, resource allocation, scaling, and handling resource failures. This distributed nature of Kubernetes, achieved through a cluster of interconnected nodes, ensures high availability and fault tolerance for applications.

Kubernetes is designed to be highly versatile, running on any scale and architecture. It operates on a variety of host machines as long as a Container Runtime Environment (CRE) like Docker and Kubernetes tools are installed, making it operating system and hardware independent. This flexibility allows Kubernetes to be deployed in diverse environments, including on-premises data centres, public clouds, or hybrid cloud setups.

DRAFT

With its container orchestration capabilities, Kubernetes streamlines the deployment process, making it faster and easier. Updates and changes can be rolled out with minimal downtime, ensuring continuous availability of services. Kubernetes also can detect and restart failed containers, ensuring the resilience of applications.

Another benefit of Kubernetes is its ability to load balance traffic across multiple containers, ensuring optimal resource utilisation and scalability. By appropriately scaling the nodes, Kubernetes prevents application failures due to resource constraints. Additionally, Kubernetes provides the ability to mount and add storage, allowing the deployment of stateful applications that require persistent data storage.

The core building blocks of Kubernetes include Pods, which are groups of one or more containers with shared storage and network resources. Deployments define a set of pods using a template and replica count, specifying the desired number of pods to run. Services provide a stable endpoint to direct traffic to the desired pods, even as they change dynamically. Ingress objects expose application endpoints to external traffic, typically through HTTP.

Kubernetes is designed to enable communication between pods without the need for NAT (Network Address Translation), which poses a challenge when dealing with remote nodes. As it is already mentioned above, the MARVEL E2F2C framework includes such remote nodes in edge and fog layers that belong to independent networks. NAT is necessary for the communication of all these individual networks. This issue can be resolved by using a VPN (Virtual Private Network). By establishing a VPN connection, all participating nodes are brought together as if they were part of the same local network. This eliminates the complications of NAT or firewalls and ensures seamless communication between the nodes. As a result, all components deployed in Kubernetes can utilise the VPN tunnel for their communication needs. This ensures a smooth and transparent network environment for the entire Kubernetes infrastructure.

MARVdash

MARVdash is a user-friendly Kubernetes dashboard designed to simplify the instantiation of orchestrated container services in the MARVEL E2F2C framework. It enables domain experts to interact with platform resources without requiring in-depth knowledge of lower-level tools and interfaces. By serving as the landing page for users, MARVdash acts as a gateway to MARVEL's E2F2C framework, allowing users to launch services, design workflows, request resources, and specify execution parameters through an intuitive web-based interface.

MARVdash itself is deployed as a service on Kubernetes, providing a graphical interface for managing services and applications launched from customisable templates. It ensures secure access to multiple services through a single externally accessible HTTPS endpoint. Additionally, MARVdash includes a private Docker registry for organising container images. User management is handled at a high level, with services assigned to individual namespaces to ensure isolation. Furthermore, MARVdash enables seamless interaction with datasets, automatically attaching them to service and application containers upon launch.

It is important to note that MARVdash does not perform processing itself. Instead, it facilitates the efficient deployment of services across various computing layers that support container-based execution. To configure and initiate services, MARVdash utilises a service templating mechanism. Each service is defined with variables that can be assigned values by the user as execution parameters through the dashboard. MARVdash takes care of setting other "internal" platform configuration values, such as the location of the private Docker registry and external DNS name, among others.

DRAFT

3.2 Deployment optimisation

3.2.1 MARVdash new deployment method

In the first version of MARVdash, users could deploy AI and other MARVEL components on individual nodes within the Kubernetes cluster. The component owners had the option to select a specific cluster node for deployment based on their requirements. To achieve this, a combination of taints, tolerations, and affinity rules were employed to dedicate nodes exclusively for specific pods (each MARVEL component or application corresponds to a set of pods).

The utilisation of taints and tolerations ensures that pods without appropriate selectors are not placed on tainted nodes. This prevents incompatible pods from running on those nodes. Additionally, node affinity is employed to prevent pods with specific tolerations from being scheduled on unlabelled nodes, further enforcing deployment restrictions. These Kubernetes mechanisms are thoroughly explained in D3.2.

However, the decision regarding the execution environment of a component was made manually. This involved modifying the tolerations and affinity attributes within the YAML file of each component. By adjusting these values, users could control the placement of pods and specify their desired execution environment within the cluster, allowing for fine-grained control and customisation while deploying MARVEL components.

The goal of the updated MARVdash is to automate the selection of deployment targets based on resource availability at each layer of the MARVEL E2F2C framework. This updated version of MARVdash simplifies the process of writing the YAML file by eliminating the need for declaring information regarding the deployment node. MARVdash users now have the option to select specific node based on the deployment layer as well as the resource requirements of their components using the MARVdash UI (i.e., GPU). In this case, the component will be deployed on a specific node based on the choice made by the user. The user can also select just the desired deployment layer, without specifying the node. In that case, the component will be deployed on any node on that specific layer that satisfies the resource requirements and constraints. MARVdash extends the provided YAMLs with the corresponding tolerations, affinity, and resource fields. Kubernetes scheduler will then make decisions regarding the desired execution environment by considering the component's resource requirements and the actual availability of devices, network resources, and overall resource consumption throughout the E2F2C framework.

The Kubernetes scheduler[8] follows a strategy known as "best-fit" when making scheduling decisions for pods. The scheduler first applies a set of filtering rules to eliminate nodes that do not meet the pod's resource requirements and constraints. This includes checking CPU and memory availability, node selectors, taints and tolerations, and other affinity rules. Once the initial set of suitable nodes is identified, the scheduler assigns a score to each node. Finally, the pod is assigned to the node with the highest ranking. Kubernetes scheduler ensures that the sum of the resource requests from the scheduled containers within the pods is within the node's capacity. Even if the actual resource usage, such as memory or CPU, on a node is currently low, the scheduler will not assign a pod to that node if the capacity check fails. This mechanism serves as a safety measure to accommodate potential resource usage peaks.

[8] https://kubernetes.io/docs/concepts/scheduling-eviction/kube-scheduler/

DRAFT

Towards the above, we implemented new fields in the Creation of each Service where the MARVdash user can easily select Layer, Node, and GPU (Figure 3) without having manually to change the YAML file. When the MARVdash user selects the layer (Figure 4) the lists with the available nodes are updated (Figure 5). The user can select whether to run the service with GPU (Figure 6) if the node has one, or not (Figure 7). If the MARVdash user selects GPU, then a list with the appropriate values (Figure 8) will appear to create the appropriate requests and limits for GPU. By changing the Layer, the list of nodes is updated (Figure 9) and also the list of available resources (Figure 10). By selecting the above, MARVdash takes care of the instantiation of the appropriate service in the correct node.

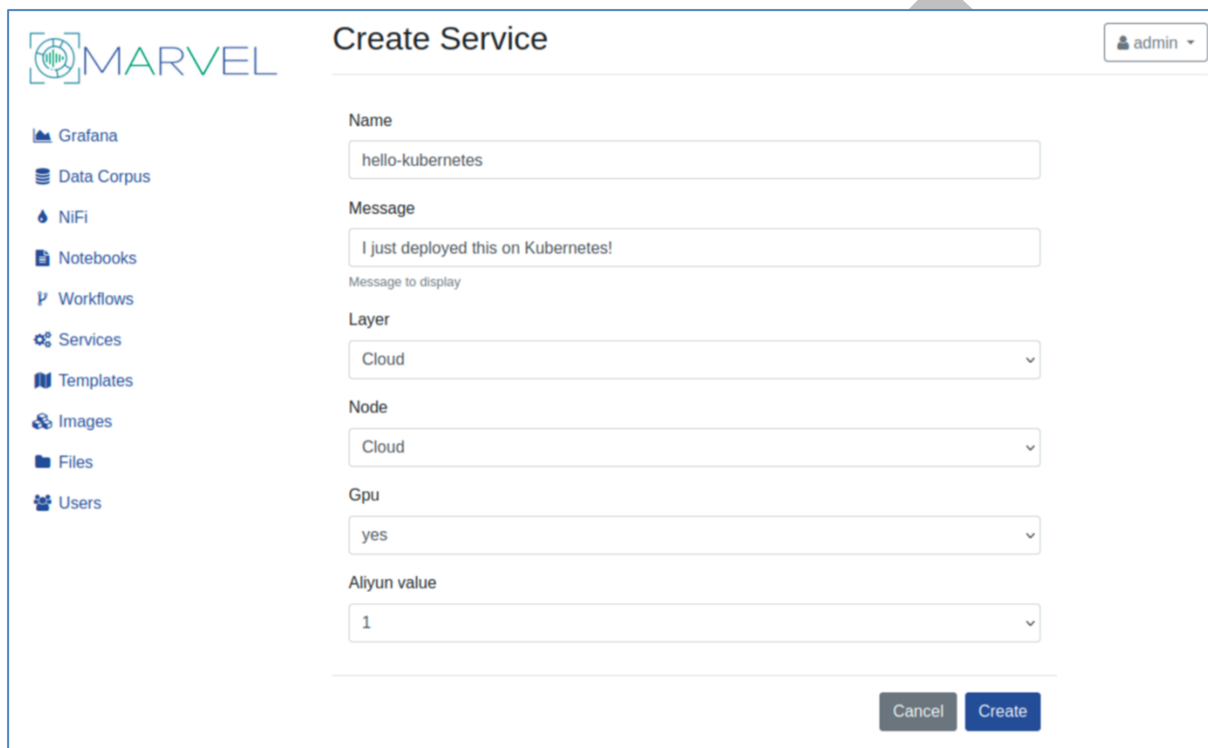

Figure 3. Create Service Updated

D

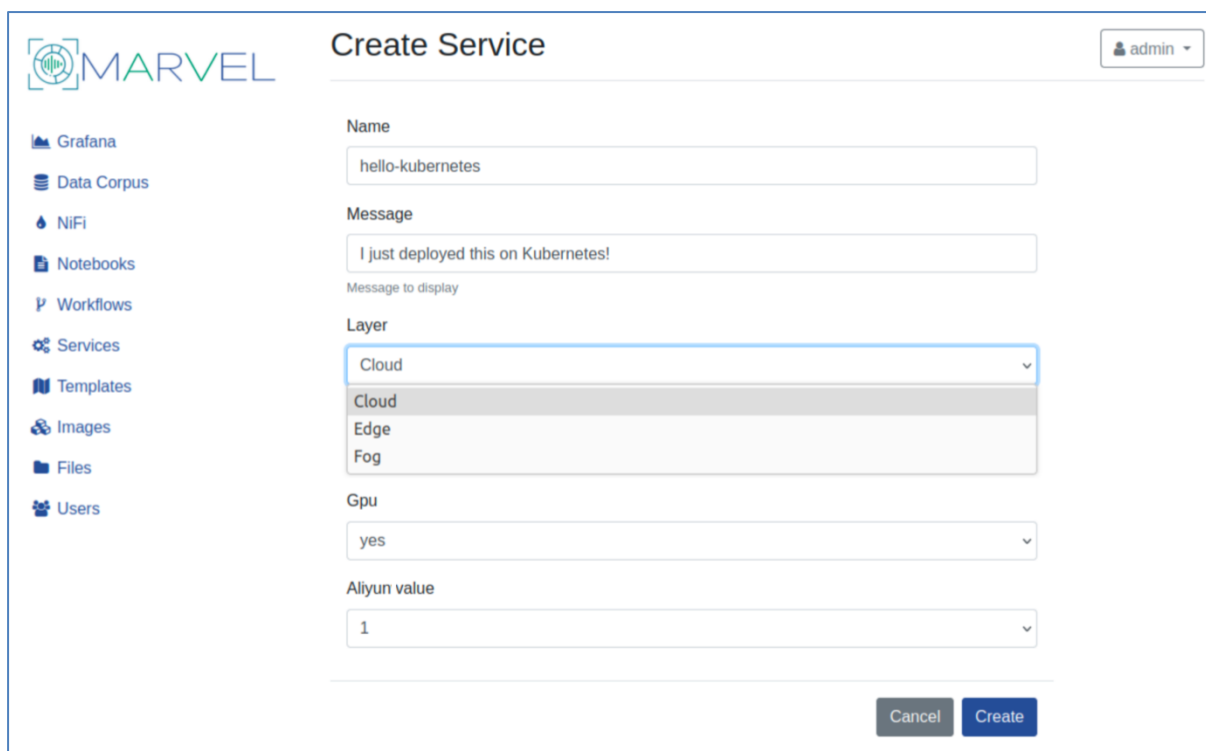

Figure 4. Create Service - Select layer

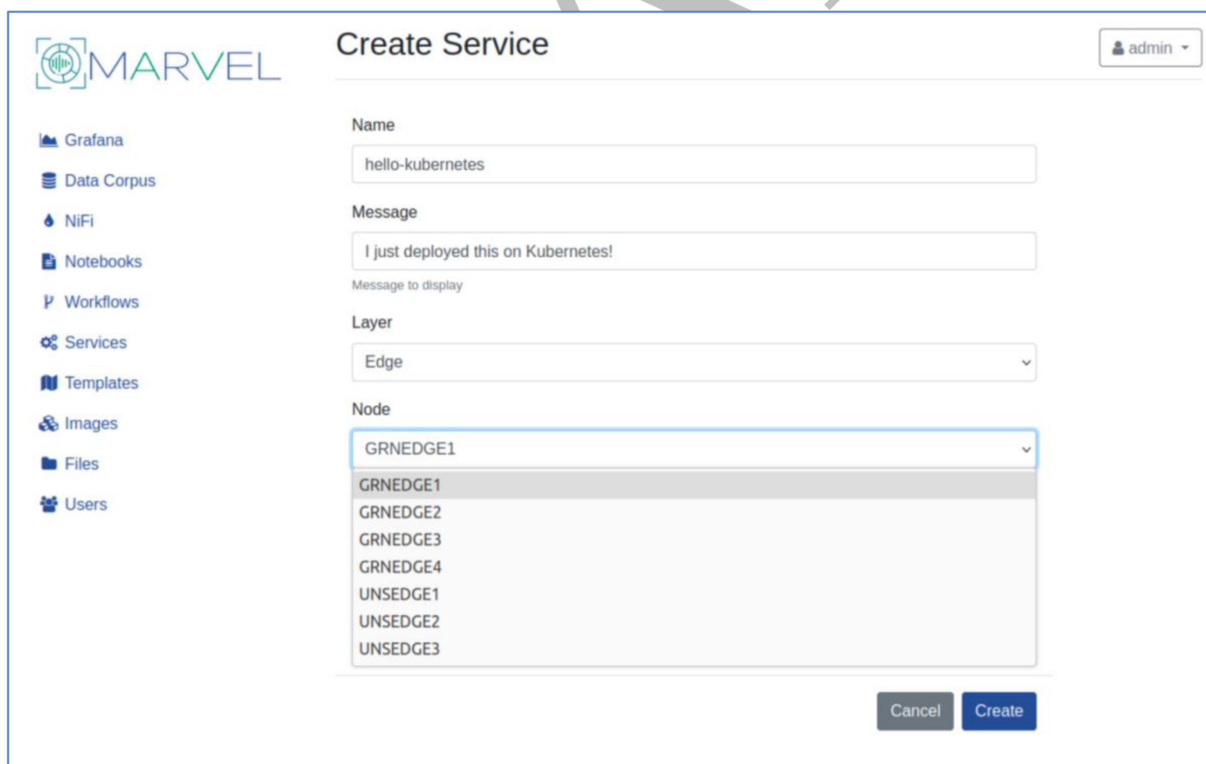

Figure 5. Create Service - Select Node

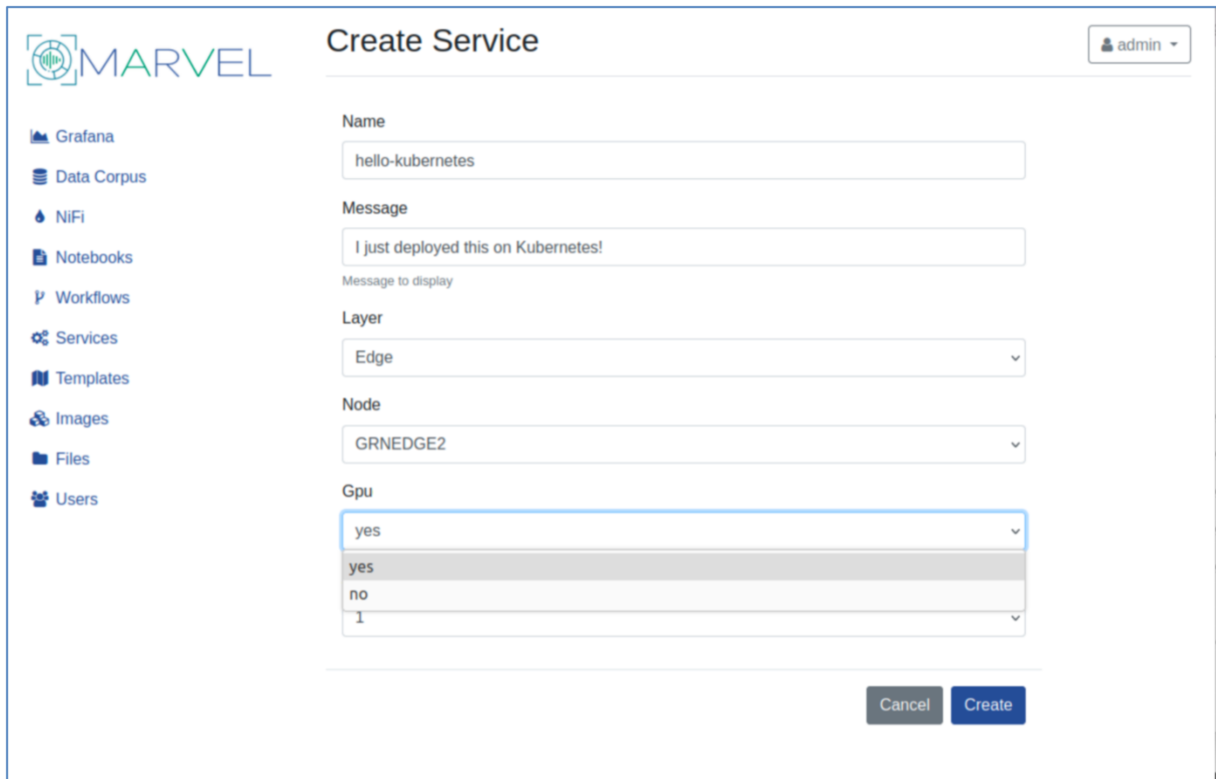

Figure 6. Create Service - Select GPU

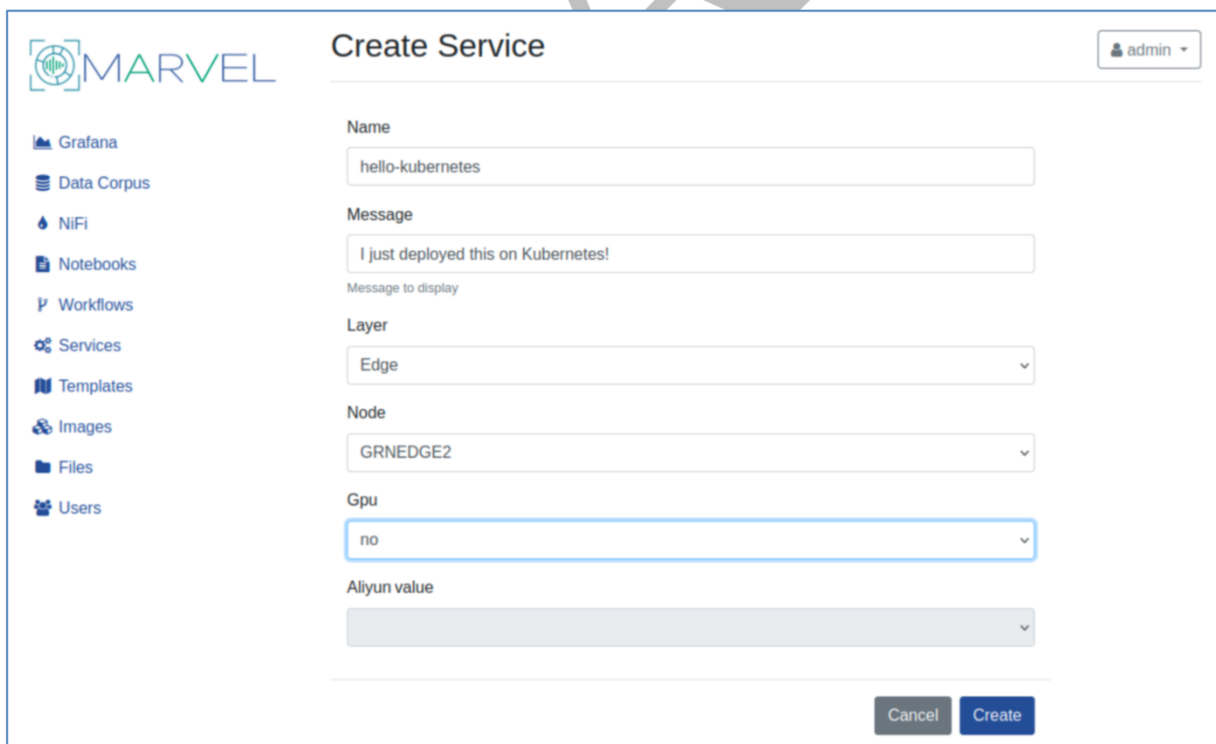

Figure 7. Create Service - Without GPU

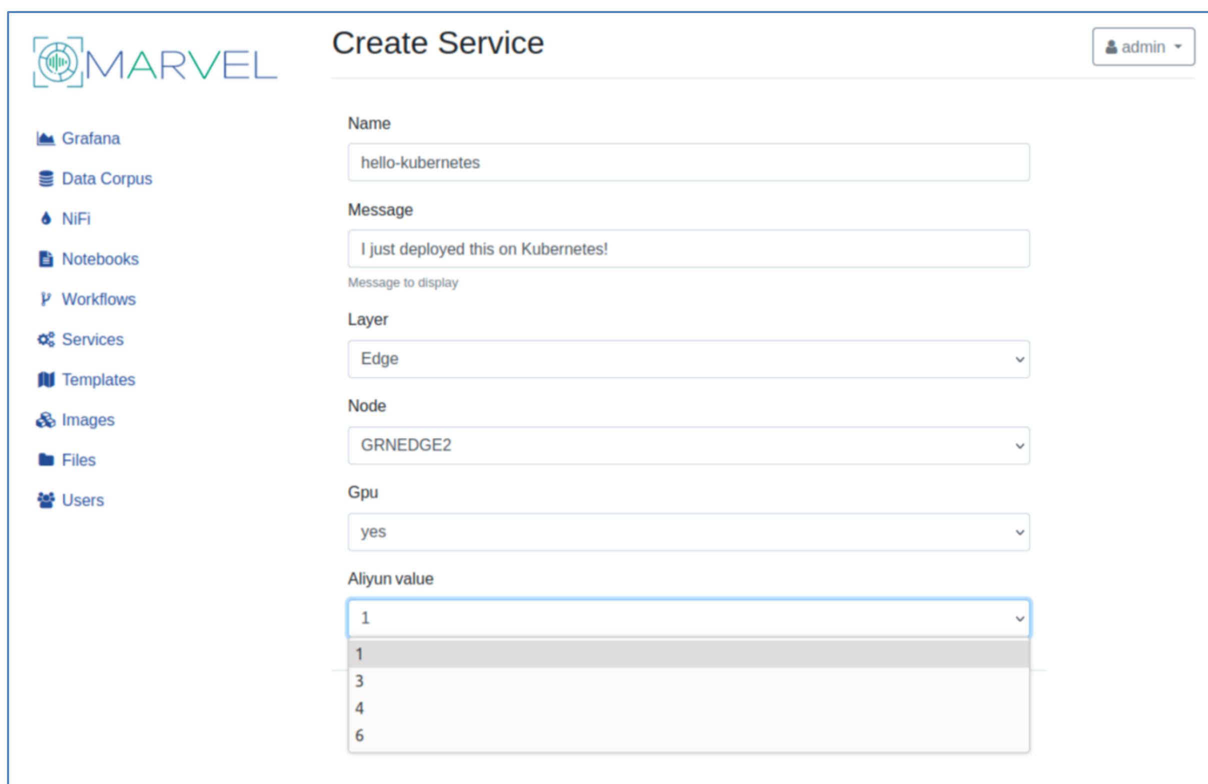

Figure 8. Create Service - GPU setting requests and limits

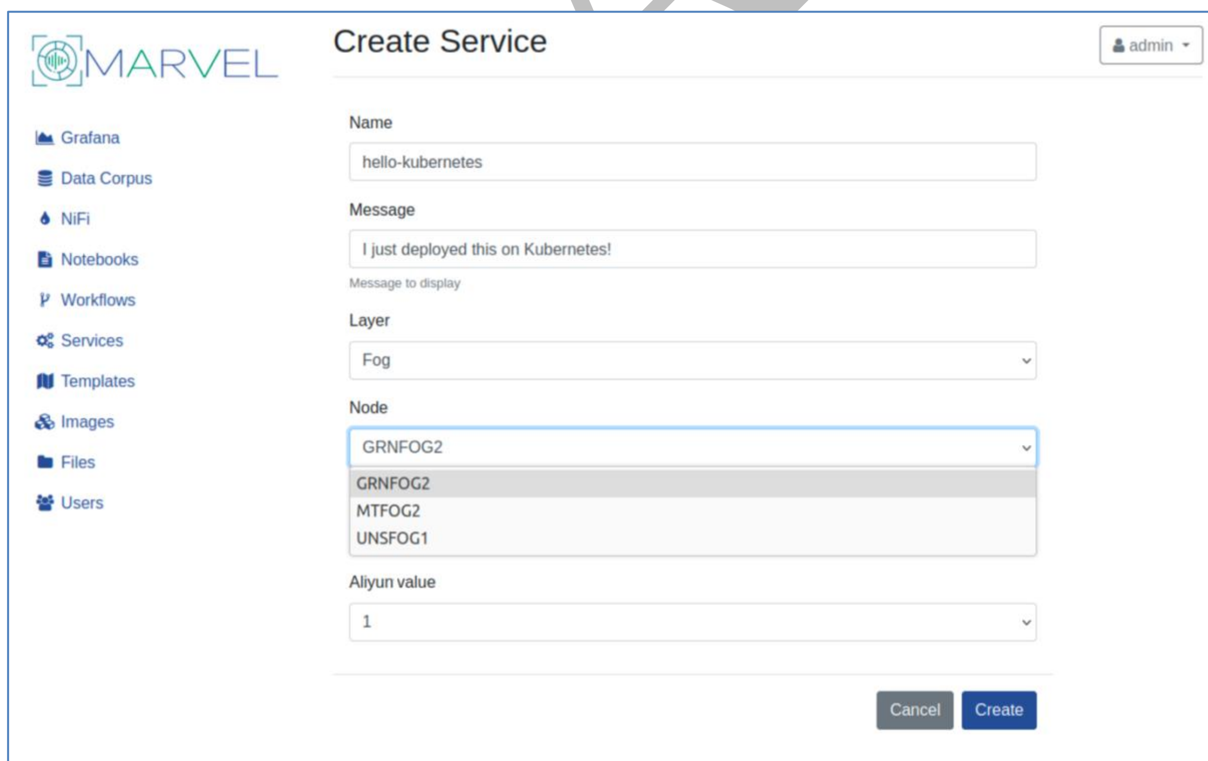

Figure 9. Create Service - Select Fog Layer

Figure 10. Create Service - Select Fog Layer Node with GPU

By implementing this approach, we achieved deployment optimisation by intelligently determining where the processing should take place, thereby optimising the distributed DL (Deep Learning) architectures within the E2F2C framework.

3.2.2 Monitoring and optimisation tools

One of the primary goals is to manage and maximise the performance of a Kubernetes cluster. Towards this goal, we have implemented monitoring tools. These tools provide real-time visibility into the health and performance of a Kubernetes cluster, enabling administrators to proactively identify and resolve issues. By this, admins ensure high availability and reliability of applications running on the cluster. By leveraging these tools, MARVdash admins can make data-driven decisions to improve efficiency, reduce costs, and enhance the overall performance of their Kubernetes infrastructure.

To serve the above purpose, we installed in the MARVEL's Kubernetes cluster Prometheus, Grafana, Loki and Zabbix.

Prometheus[9] is an open-source monitoring and alerting system specifically designed for monitoring highly dynamic and distributed environments. It provides a robust infrastructure for collecting, storing, and analysing metrics from various systems, applications, and services in real time. One of Prometheus's main features is a multi-dimensional data model with time series data identified by metric name and key/value pairs. Another key feature is its flexible and powerful query language, PromQL, which allows users to slice, aggregate, and analyse

[9] https://prometheus.io/

collected metrics. In addition, Prometheus has no reliance on distributed storage; single server nodes are autonomous and time series collection happens via a pull model over HTTP. Pushing time series is supported via an intermediary gateway and targets are discovered via service discovery or static configuration Prometheus's architecture is designed to be highly scalable and resilient. It can be easily integrated with other monitoring tools and systems through its extensive range of exporters, enabling seamless monitoring of various components within a system or infrastructure.

Grafana[10] is a popular open-source data visualisation and analytics platform known for its rich features and user-friendly interface. It is widely used to create interactive dashboards that help users gain deep insights from their data. With Grafana, users can connect to a variety of data sources, including databases, monitoring systems, and cloud services, and easily visualise the data in the form of charts, graphs, and tables. Grafana has an extensive library of pre-built panels and plugins. This allows users to customise and design dashboards tailored to specific needs. It supports real-time data streaming and dynamic updates, enabling users to monitor metrics and track changes as they happen. Moreover, Grafana's intuitive query editor and powerful query language give the ability to users to perform advanced data analysis and exploration.

Grafana complements Prometheus since it integrates seamlessly with Prometheus to retrieve and display the collected metrics in visually appealing dashboards. More specifically with Grafana, MARVdash users can create customisable dashboards, charts, and graphs to visualise the performance and health of the Kubernetes cluster. Prometheus collects and stores metrics, while Grafana offers a user-friendly interface to explore and visualise the collected data. This combination allows MARVEL administrators and developers to gain real-time insights into the cluster's performance, troubleshoot issues, and make data-driven decisions to optimise the MARVEL E2F2C framework for better reliability and efficiency.

On top of the above, there was a need in the MARVEL framework for users to have access to Logs apart from the ones provided by MARVdash through the instantiation of Kubebox from the templates of MARVdash. As a solution to this need, we installed ***Loki***[11] in the Kubernetes cluster. Loki is an open-source log aggregation system developed by Grafana Labs. It is designed to provide efficient and cost-effective log storage and analysis for modern distributed architectures like Kubernetes. Instead of following the traditional centralised log storage approach, Loki utilises a unique indexing strategy called "labels". This approach allows logs to be indexed based on user-defined labels, enabling fast and efficient querying and retrieval of log data. This way Loki facilitates quicker log searching and analysis. Loki integrates smoothly with Grafana, enabling MARVdash users to create visualisations and dashboards based on log data. Loki simplifies log management, streamlines troubleshooting processes, and empowers organisations to derive valuable insights from their log data.

MARVEL users can navigate to the pre-built dashboards as well as to the custom-made dashboards which visualise the MARVEL environment. In each node of the Kubernetes cluster, there is a pod of Prometheus collecting the information and this is visualised through Grafana which is accessible via the MARVdash menu (Figure 11) to all users.

[10] https://grafana.com/

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

DRAFT

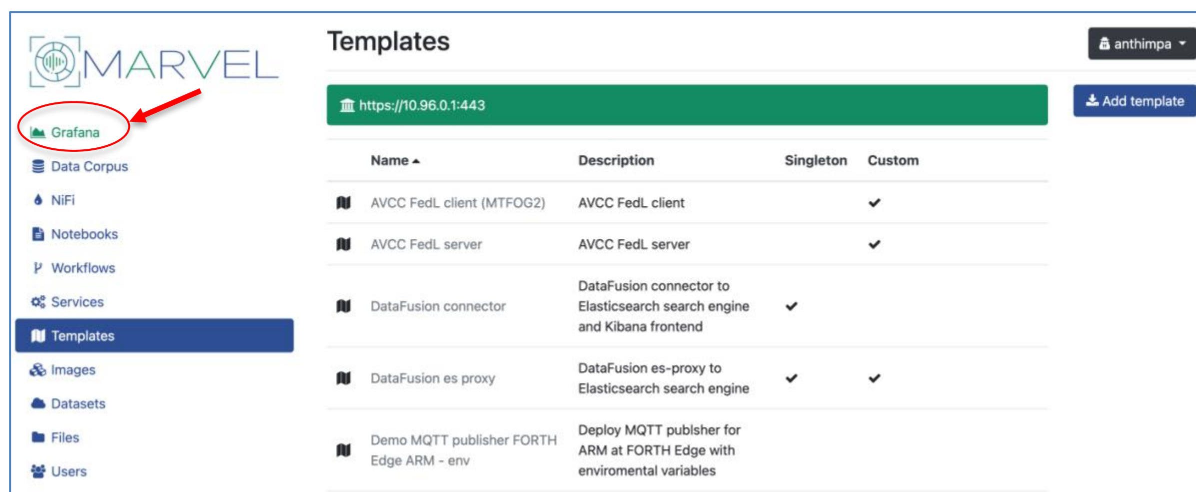

Figure 11. MARVdash updated menu

By selecting the "Grafana" option, the MARVdash user can select from a list of dashboards accordingly (Figure 12).

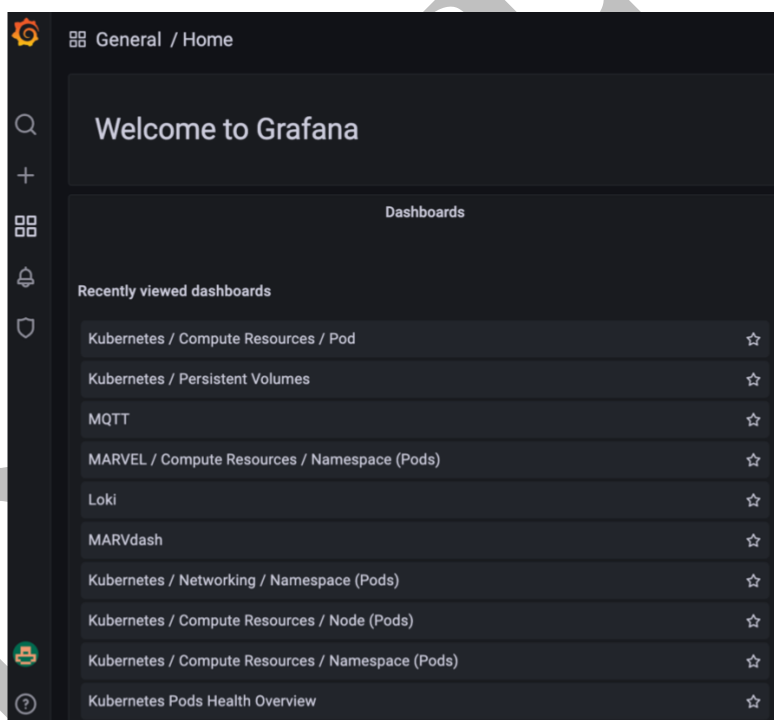

Figure 12. Grafana Dashboards

By selecting for example one of the pre-built dashboards Compute Resources per namespace (Figure 13), the MARVdash user can see all the pods that are instantiated and the metrics for each one of them. Some of the metrics that are displayed are CPU Quota, Memory Quota, Memory Usage, Current Network Usage, Bandwidth, Rate of packets, Rate of Packets Dropped, and Storage IO. Moreover, through Grafana, the user can access the dashboard MARVdash (Figure 14), where the logs of the pods concerning the MARVdash deployment are depicted. The MQTT dashboard is also a very useful one since logs from MQTT brokers that are instantiated through MARVdash in the Kubernetes cluster are visualised (Figure 15).

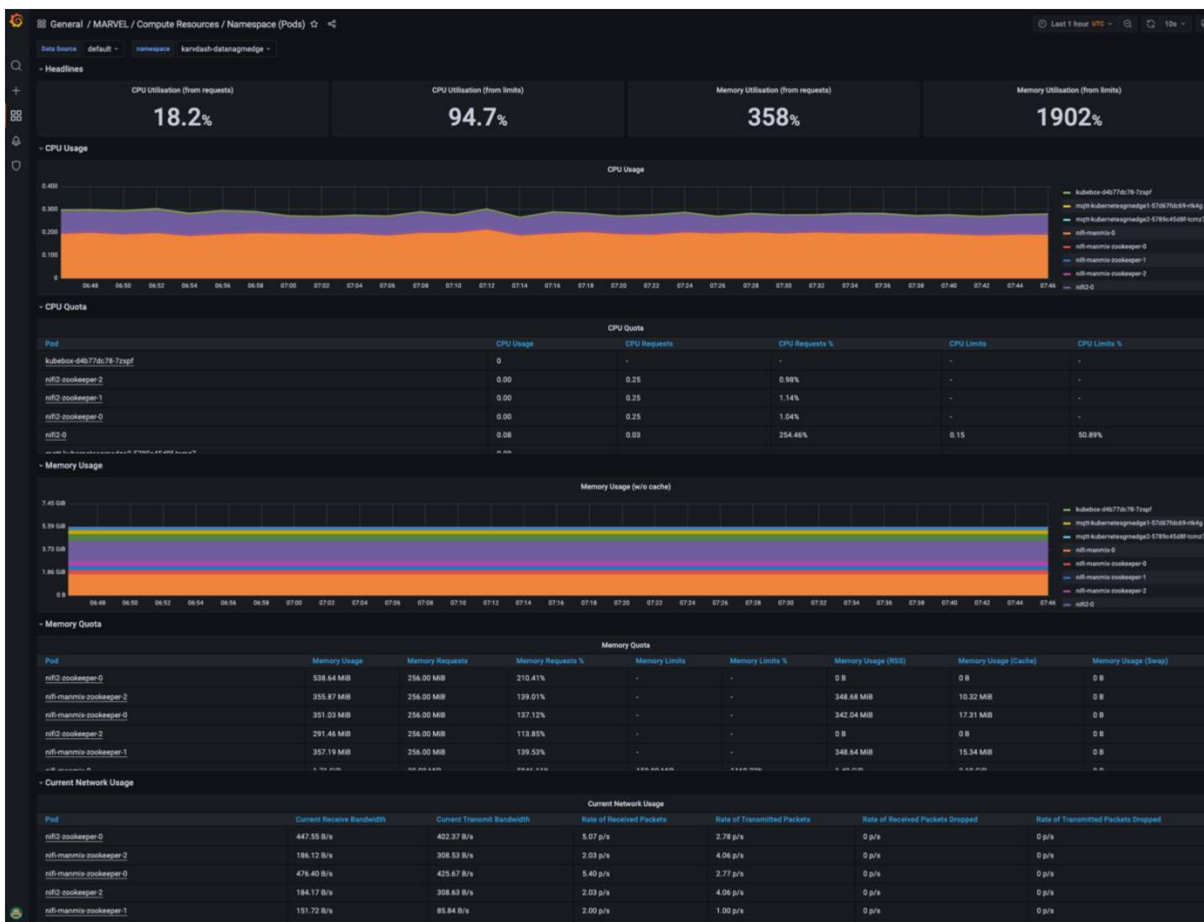

Figure 13. Compute Resources per namespace

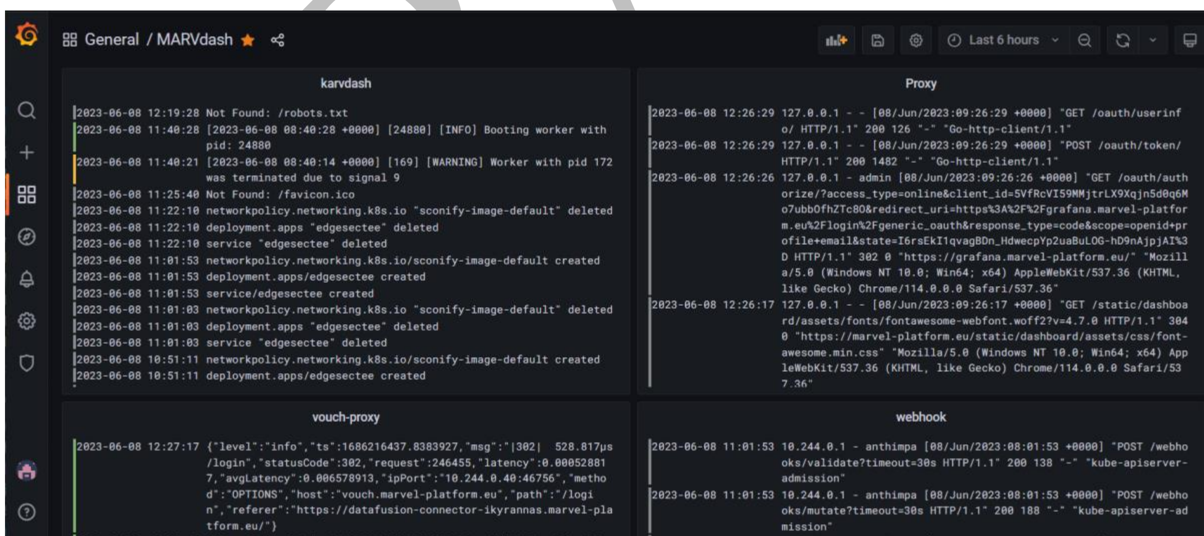

Figure 14. Grafana MARVdash dashboard

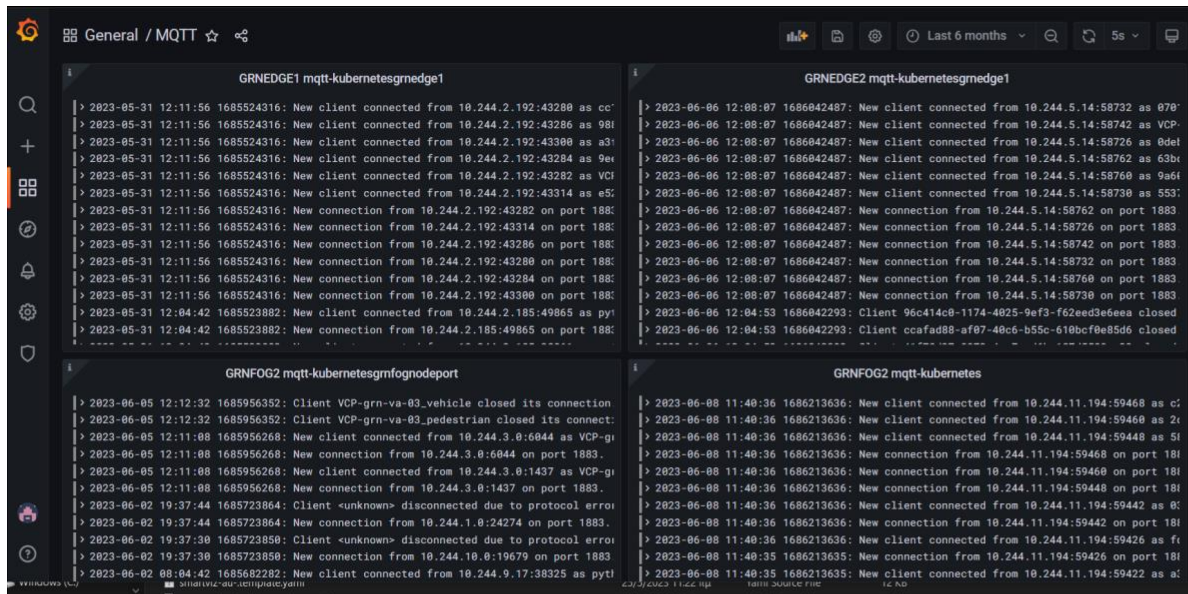

Figure 15. Grafana MQTT dashboard

To strengthen the monitoring options in MARVEL's Kubernetes cluster we installed ***Zabbix***[12] agents on each node. Zabbix is a widely used open-source monitoring solution that provides comprehensive monitoring, alerting, and visualisation capabilities. Zabbix allows users to monitor various aspects of their IT infrastructure, including servers, networks, applications, and services. It supports both agent-based and agentless monitoring approaches, giving flexibility in monitoring different types of devices and systems. Zabbix, offers real-time monitoring of performance metrics, such as CPU usage, memory utilisation, network traffic, and disk space, providing administrators with valuable insights into the health and performance of their infrastructure. It also supports advanced monitoring capabilities like event correlation, trend analysis, and anomaly detection.

Zabbix provides a user-friendly web interface where users can configure monitoring settings, create dashboards, and generate reports for analysing historical data. It also supports the creation of custom templates, making it easy to monitor specific applications or devices with pre-defined configurations. For the sake of MARVEL, up to now, we have created 2 custom dashboards. One dashboard to visualise the basic metrics (CPU, RAM, GPU, DISK USAGE) of each node of the Kubernetes cluster (Figure 16) and one for the COPRUS (Figure 17).

DRAFT

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

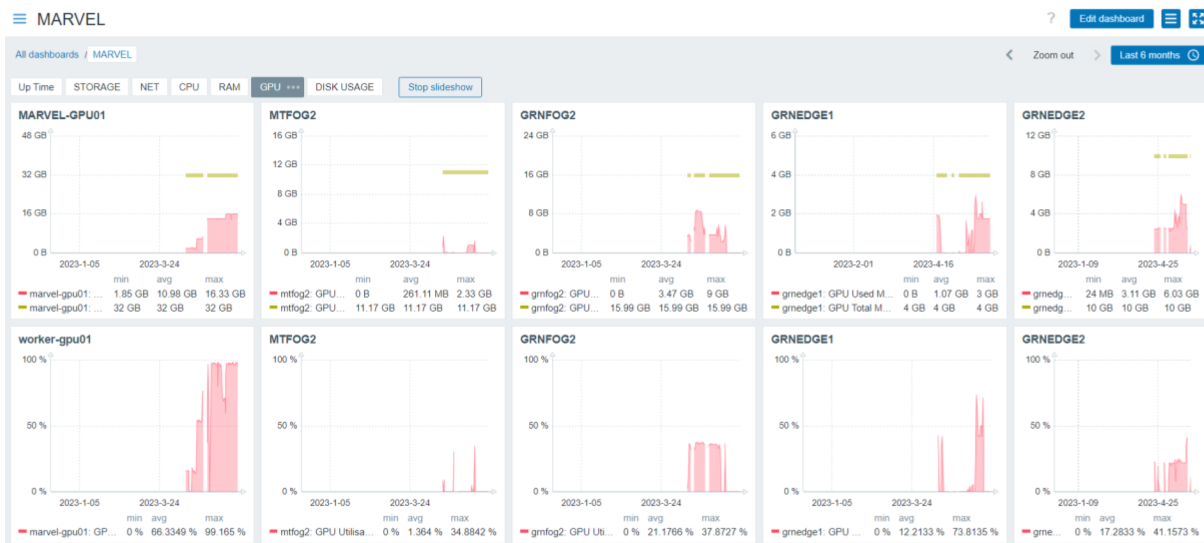

Figure 16. Zabbix MARVEL dashboard

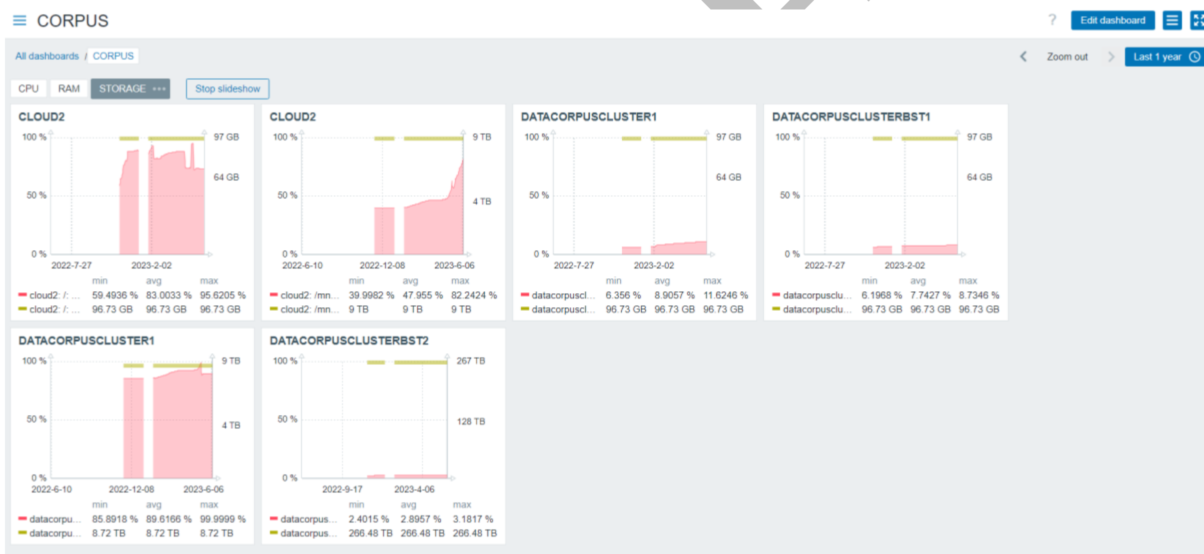

Figure 17. Zabbix Data Corpus dashboard

3.2.3 Real-time decision-making in all layers of the MARVEL E2F2C framework

Real-time decision-making refers to the process of making informed and timely decisions based on up-to-date data and insights. In the context of E2F2C computing, several exit points exist where decisions can be made based on the data flow and processing stages. Edge computing refers to processing and analysing data closer to the source, typically at or near the devices or sensors generating the data. At the edge, decisions can be made instantaneously, allowing for low-latency response times and reduced dependence on cloud connectivity. Fog computing, which occurs between the edge and the cloud, involves distributing computing resources and data storage in a decentralised manner. At the fog layer, decisions can be made to filter and aggregate data, perform preliminary analysis, and prioritise information before transmitting it to the cloud. The cloud, as the central repository for data storage and processing, provides a powerful and scalable infrastructure for advanced analytics and decision-making. At the cloud

level, data can be processed holistically, leveraging machine learning algorithms and data models to derive deeper insights and make complex decisions.

By leveraging these exit points within the E2F2C framework, MARVEL users can make decisions at various stages of data processing. This distributed decision-making approach allows for a balance between local autonomy and centralised control, by enabling optimisation of operations, improving efficiency, and enhancing overall decision-making capabilities.

To serve the above purpose, we need to be able to instantiate or delete the service that is responsible for making decisions at different layers. A REST API is provided by MARVdash to perform service management from external systems. This API includes the following methods (Table 3) under the API's base URL[13]:

Table 3: MARVdash API methods

Method	Path	Description
GET	/services/	List running services
POST	/services/	Create/start a service
POST	/services/<name>/	Execute a command at service pods
DELETE	/services/<name>/	Delete/stop a running service
GET	/templates/	List available templates
POST	/templates/	Add a template
GET	/templates/<id>	Get template data
DELETE	/templates/<id>	Remove a template
GET	/services/	List running services
POST	/services/	Create/start a service
POST	/services/<name>/	Execute a command at service pods

All methods use a JSON dictionary for input data and respond using JSON formatting. A Python library called karvdash_client has been implemented to easily use the API in any Python script.

To use the client library, you should provide the API endpoint and an authentication token in a configuration file. For MARVdash users this file is automatically created and mounted in pods at /var/lib/karvdash/config.ini.

The output of the GET /services/ is a dictionary containing the following keys (Table 4):

Table 4: Output of GET services

Key	Type	Description
name	<string>	The service name
url	<string>	The URL to access the service frontend
created	<string>	When the service was created (not included in response to service create calls)
actions	<boolean>	True if service can be deleted
template	<dictionary>	Service template information

The output of the GET /templates returns a dictionary with the following keys (

[13] https://marvel-platform.eu/static/docs/api.html

DRAFT

Table 5):

Table 5: Output of GET templates

Key	Type	Description
id	<string>	The template identifier
name	<string>	The template name as shown in the dashboard
description	<string>	The template description as shown in the dashboard
singleton	<boolean>	True if only one instance can be running
auth	<boolean>	True if HTTP authentication should be added by the ingress
datasets	<boolean>	True if dataset volumes should be mounted in pods
variables	<dictionary>	Template variables
values	<dictionary>	Instance values for template variables (included when template is returned as part of a service)
filename	<string>	The template filename (included for system templates)
data	<string>	The actual template (included when requesting a single template)

To start a service, you should use as parameters the following:

- identifier (string) – the template identifier
- variables (dictionary) – template variables as key-value pairs (provide at least a name key)

and returns: The service created.

To delete a service, you should use as parameters the following:

- name (string) – the service to delete

All the above were realised in DMT. This functionality is made possible through the interaction between SmartViz and MARVdash via its API (Table 3). By utilising this functionality, SmartViz can communicate with MARVdash and retrieve information about the services that are currently available and deployed within the system. Users can then initiate and terminate services on specific exit points (edge, fog, cloud). For example, in Figure 18 the user can terminate the SED component from the edge and initialise it to a different exit point such as fog or cloud. This interaction provides the end users with the ability to control and manage services within the system. Therefore, an efficient distributed E2F2C DL deployment is realised enabling real-time decision-making in all layers of the framework.

DRAFT

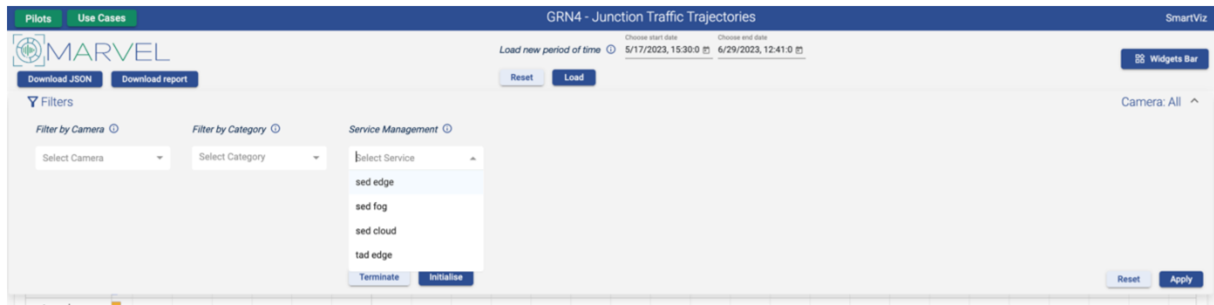

Figure 18. SmartViz Service Management

DRAFT

4 Model optimisation for efficient Inference

4.1 Methods and approaches for efficient Inference

The DynHP library has been developed using the following frameworks: Pytorch and Pytorch-Lightning. This ensures that the application of DynHP to a generic model, built using both densely connected and convolutional layers, remains modular.

4.1.1 Compression applied to Audio Visual Crowd Counting Models

The DynHP has been applied to the AVCC architecture developed within MARVEL. In this architecture, there are many options regarding where to apply the gates used by DynHP to identify the groups of parameters that can be safely removed. Given the specific purpose of AVCC, i.e., the production of heatmaps used to count the number of people in a specific scene (Figure 19), removing entire filters for reducing the size of the model might be critical.

The specific procedure adopted in this case is to apply the gates only to the convolutional layers in the Fusion Blocks and to perform the standard train through DynHP. To maintain consistency with the two input branches and the output format, the first and last blocks of the Fusion Blocks are left unchanged. Once done, the resulting network has been re-instantiated in the compressed form and trained a second time using 16 bits precision.

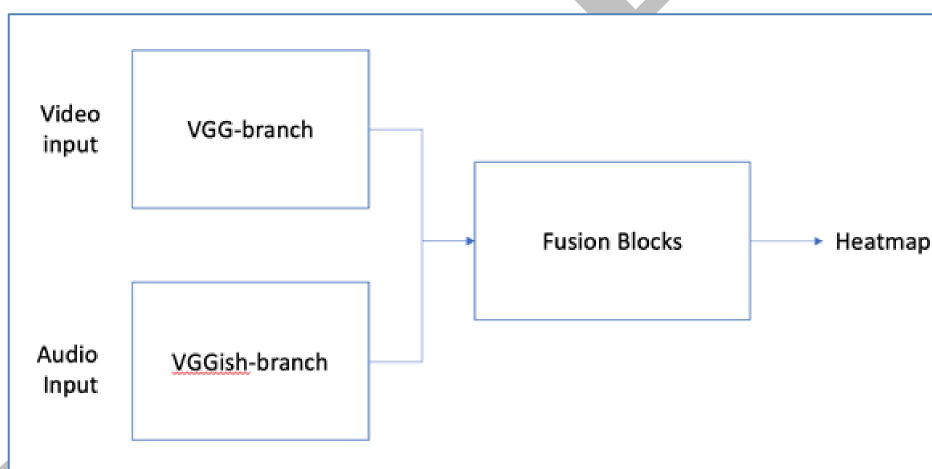

Figure 19. Production of heatmaps

The results are showcased in the following. Figure 20 shows an example of input taken from the Disco dataset used to test the performance of the methodology. We compare the output of the compressed model with two benchmarks: the uncompressed AVCC model in full precision (32-bit) and the same uncompressed model trained with half-precision (16-bit).

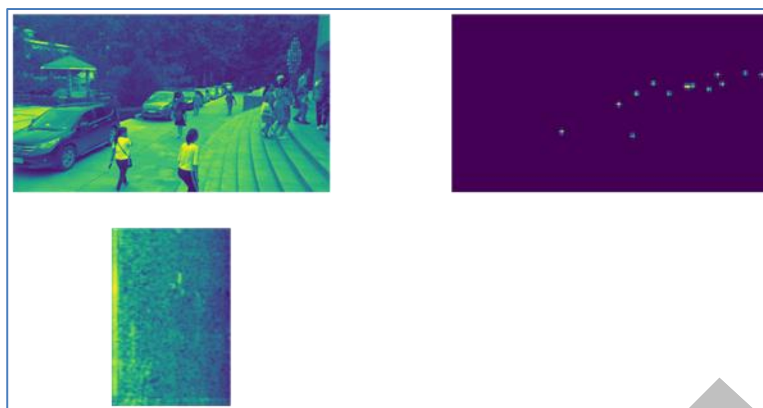

Figure 20. Example of A/V Input Output from Disco dataset

As shown in Table 6 the compression reduces the overall size of the model by 60% limiting the accuracy degradation limited to 3.14%. As reported in the last row, the compressed model is 58% faster than the original one.

Table 6: Compression-accuracy performance comparison between compressed vs. uncompressed models trained on Disco Dataset

	AVCC Full (original)	AVCC Half prec. (16bits)	AVCC Half compressed
Mean Abs Error (difference %)	18.11	17.04	21.25
Size (MB) (gain %)	88 (-)	48 (50%)	35 (60%)
Time (s) (1 frame exec time increment %)	0.48 (-)	0.32 (33%)	0.20 (58%)

4.1.2 Compression of Visual Crowd Counting Model

The DynHP compression methodology has been applied to a second AI model, i.e., SASNet, used in MARVEL for Visual Crowd Counting (VCC).

As shown in Figure 21, the SASNet architecture is a U-Net with a rather complex connectivity concerning the pruning process performed by DynHP. We recall that DynHP performs structured pruning which, in this specific case, means pruning the output channels of the Conv2d Layers used in the network. We observed that applying the pruning methodology to both encoder and decoder at the same can be too aggressive, due to the inherent structure of the SASNet architecture, whose expected output is a heatmap. Indeed, this resulted in severe damage to the capabilities of the network in reconstructing the input images into the corresponding heatmaps. Note that in the presence of low-quality final heatmaps, the accuracy of the crowd-counting would degrade significantly.

DRAFT

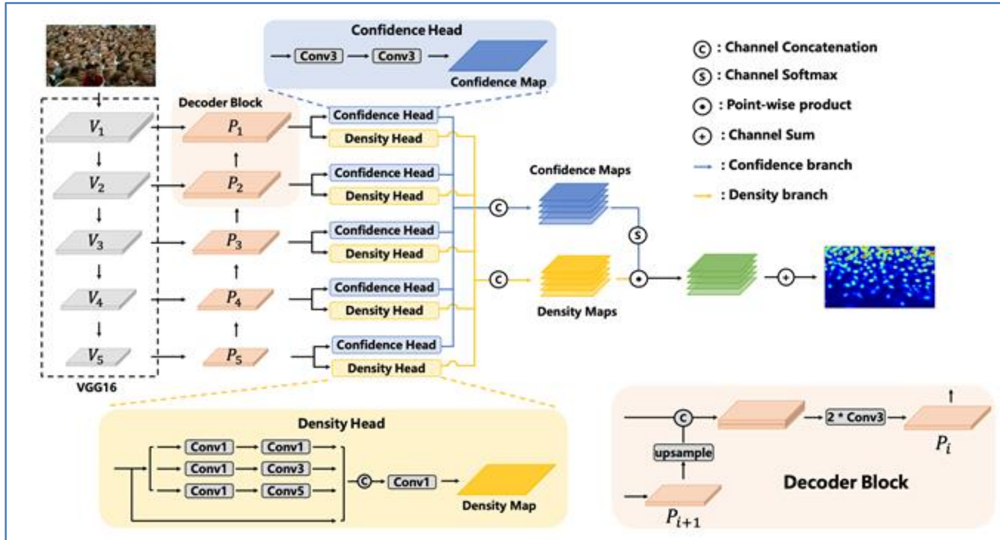

Figure 21. SASNet architecture [4]

Therefore, we applied the L0-layers, i.e., the layers equipped with the gates used to learn which groups of parameters can be removed, to the Decoder only. Placing the gates on the Decoder has the advantage of propagating the effects of pruning also to the other blocks: i.e., the Encoder and the upper layers (Confidence and Density Heads).

As for AVCC, the compression process is divided into two phases: i) training and compression and ii) fine-tuning, using half precision. The resulting compressed model on the MT dataset collected in MARVEL is reported in Table 7.

Table 7: Comparison between VCC full and compressed model

Metric	VCC Full (32bit)	VCC compressed (16bit)
Accuracy (MAE)	27	29 (-2%)
Size (MB)	148	53 (64%)

4.1.3 Deployment and evaluation of compression on an edge device

The compressed model has been deployed on an edge device for the sake of performance evaluation. The edge device adopted for such a test is an Apollo Dev Kit produced by Smartcow. Such a device is equipped with an Nvidia Jetson Xavier NX with 16GB of RAM and Nvidia Jetpack 4.6.

The deployment has been performed directly on the system, and no intermediate layers, e.g., Docker, were used.

The whole deployment procedure is as follows:

1. Train the model on a capable device. For this phase, the current version of the library is compatible with Python 3.6 and newer versions.
2. Export the final state model, i.e., using the standard exporting methods of Pytorch.
3. Move the exported model to the edge device, i.e., the Apollo Dev Kit.
4. Export the Model in TorchScript format.
5. Run the performance evaluation

Figure 22 shows how to create the Torchscript version of the compressed model.

DRAFT

```python
import torch

from models_pl.orig_models import AVCC

# chpt_full = "./apollo/avcc-full.chpt"
chpt_full = "./apollo/avcc-comp.ckpt"

model = AVCC.load_from_checkpoint(chpt_full,pretrained_backbone=False, fb_filters=[512,
256, 256, 128,64, 64])

save_path = "./apollo/avcc-disco-comp.pts"

is_half = True

model.eval()

if is_half:
    model.half()

model_script = model.to_torchscript()

torch.jit.save(model_script,save_path)

print(f'Created torchscript model in {save_path}')

```

Figure 22. Deployment procedure of AVCC on the Apollo Device

The testing procedure consisted in performing an inference task on the device and comparing the inference time of both compressed and original AVCC models. In this test, we measured only the inference time once the data is loaded into the GPU memory. Optimising the data transfer is not achievable with model compression.

Table 8: Inference time performance evaluation

Model	Inference Time	Speedup
AVCC Full	0.045s	-
AVCC Compressed	0.010s	4.5x

4.2 Efficient anomaly detection through decentralised and unsupervised learning

In the following, we report an efficient methodology for obtaining a decentralised and unsupervised anomaly detector that can be suitable in the context where edge devices have to collaborate to train an anomaly detector using unsupervised data.

4.2.1 Motivation

The ability to obtain valuable insights from Big Data through AI techniques is a key component in improving the services provided by a smart city to its citizens. In this scenario, resource-limited edge devices gather, process, and utilise data generated during their operations. These devices have built-in computing and communication capabilities for AI tasks.

The prevalent AI paradigm is centralised, where data gathered by edge devices is transmitted to the cloud for AI model training. However, due to the explosive growth of data generated at the edge and heightened concerns about data privacy and ownership, there is a shift towards relocating AI processes to the edge, transitioning the paradigm towards a more distributed or decentralised approach.

DRAFT

Federated learning (FL) has emerged as an attractive method for training machine learning models without the need to share raw data among different clients. This decentralised framework maintains data privacy, reduces communication overhead, and facilitates more scalable training. Challenges in federated learning encompass managing heterogeneous data distributions among clients and devising efficient solutions.

This study concentrates on unsupervised federated learning, specifically targeting the enhancement of federated anomaly detection for mobile edge devices. We examine and refine a federated anomaly detection method as presented in D3.2 that employs global information to boost performance while minimising communication overhead. Our approach is not only suitable for preserving privacy and addressing network resource constraints but also designed for utilising tiny ML models on individual nodes.

Unlike the reference work, our focus lies on analysing communication costs and selecting suitable models to attain competitive results with reduced overhead. By opting for the appropriate model architecture on the Fashion-MNIST (Modified National Institute of Standards and Technology) dataset, we successfully improve the methodology's performance, achieving an 83.33% reduction in communication cost. This makes it a highly effective solution for settings involving large local datasets and a moderate number of clients, offering a robust alternative to conventional centralised methods.

4.2.2 Methodology description

What is summarised in the following has been extensively reported in D3.4[14].

We study a distributed learning system with clients M and data distributions C, where $|C| \leq |M|$. Each client obtains a fraction d of its samples from $Cout \in C \setminus Cin$, with $Cin \neq Cout$, and the remaining (100 – d)% from $Cin \in C$. Typically, $d \in [5\%, 15\%]$ is considered realistic and is commonly used in anomaly detection contexts. The methodology's goal is to create consistent groups of clients and perform standard federated learning within those groups. In the following, we revisit the two-phase structure that constitutes the entire process.

Phase I aims to enable clients to join a group with the same (or similar) majority class Cin. Clients train a lightweight anomaly detection model on local data, and pairs of clients exchange models to classify their local data. If nodes have similar inlier/outlier ratios using each other's models, they share the same inlier class and should be in the same group. An undirected graph is generated using candidate groups from each client, and a community detection algorithm is applied to identify groups of nodes for the upcoming standard FL step.

In phase II, k groups (or communities) G_0, ..., G_k are formed. For each group, federated learning is initiated using autoencoders as models. Autoencoders are suitable because they naturally fit the FL framework and can effectively be used in AD tasks. The Federated Averaging (FedAvg) protocol is used for FL. At the end of each communication round, the trained autoencoder is shared among the clients of the same group.

4.2.3 Cost analysis for phase I

First, each client trains a model on its local data and exchanges it with the other clients. Given $|M|=n$ clients and letting S_m be the size of a single local model, the total communication cost for exchanging models between all pairs of clients can be estimated as $n(n-1) \times S_m$. Clients then

[14] MARVEL D3.4 - MARVEL's federated learning realization, 2023. https://doi.org/10.5281/zenodo.7543936.

share pairwise association information, adding a cost of n(n-1) × S_r, with S_r = 1 bit. Clients share candidate groups with a single client, who builds the graph, runs the community detection algorithm, and sends back the community information. A predefined policy selects the client for this task, like the one with the lowest ID. Assuming the average candidate group size is G_m and each client ID size is S_{id}, the total communication cost for sharing candidate groups with the selected client can be estimated as (n-1) × G_m × S_{id}. G_m can vary depending on the performance of the first step. In the worst case, G_m is equal to the number of clients, i.e., $|G_i|$ = n. In the ideal case, G_m is equal to the average size of the real groups, where each group is a set of nodes sharing the same data distribution. Assuming that the average number of nodes in each group is $p \geq 1$, it holds that in the Ideal case G_m = p = n/|C|.

In our experiments, we observed that G_m typically aligns with this ideal condition, demonstrating our methodology's effectiveness in grouping clients with similar data distributions. After the community graph has been computed, an additional cost of (n-1) × G_m × S_{id} is required for sending community information back to the clients. Therefore, the overall communication cost for the first phase can be summarised as:

$$P_1 = n(n-1)S_m + n(n-1)S_r + 2(n-1)n|C|S_id$$

The first phase's communication cost is dominated by the quadratic term $n(n-- 1)$. As the number of clients grows, this impacts the cost significantly. The model size, S_m, also becomes a dominant component in the cost compared to other information types. Thus, the first phase's communication cost can be expressed as $O(n2S_m)$, emphasising the model size's importance in the overall cost.

4.2.4 Cost analysis for phase II

In the second phase of the methodology, each group (community) G_0, …, G_k starts a federated learning instance using a corresponding model U_0, …, U_k (they all have the same architecture). The communication cost analysis in this phase involves local model updates, model aggregation, and global model update distribution. For simplicity, we first consider a single group and an external aggregator. Each client trains their model on local data and computes updates. The size of these updates depends on the model architecture, and we denote it as S_u. Clients share local updates with the aggregator, which combines these updates using the Federated Averaging algorithm.

The communication cost for sending local model updates is $|G_i|$ × S_u for each group G_i, here $|G_i|$ is the number of clients in group i. Afterwards, the aggregator distributes the global model update to all clients in the group, with a communication cost of $|G_i|$ × S_u. The total communication cost for the second phase is the sum of the costs for all groups. Given r communication rounds in each FL instance and assuming there are k groups, the formula for the total communication cost in the second phase is:

$$P_2 = rk \sum_i 2|G_i|S_u$$

Considering that the sum over $|G_i|$ counts all the clients in the system, and that selecting a client within the group as the aggregator slightly improves the communication cost (i.e., that client does not need to send its update), we can rewrite the total communication cost formula for the second phase as follows:

$$P_2 = 2r(n-k)S_u$$

DRAFT

Where n is the number of clients. Asymptotically, the communication cost in the second phase is linear concerning the number of clients and the size of the local model updates

4.2.5 Experimental setup

We evaluate the proposed methodology using a common benchmark (Fashion-MNIST) and we focus on the communication aspect. The experimental setup is the following: the Fashion-MNIST dataset has ten classes ($|C| = 10$). We ensure that clients have numerically balanced and disjoint datasets. We set $p = 9$, representing the number of clients within the same data distribution (class). The ideal partitioning that we aim to find consists of $k = |C| = 10$ groups with p clients each. We only consider the case of $p = 9$ for this evaluation.

The models used in phase I and II are convolutional autoencoders with 5 and 7 layers, corresponding to 4k and 28k parameters in total, respectively.

The results presented in the following regard: i) the correctness of the group detection, i.e., the ability of each device to group with the other devices holding data with a compatible distribution.

4.2.6 Results of Group Detection

In Table 9, we observe that most of the communities found for Fashion-MNIST consist of clients with the same majority class, such as G_0 with I_6, G_1 with I_8, and so on. An exception is given by G_6, which is formed by clients with majority classes I_2, I_0, I_3, and I_4. This indicates that there is a higher degree of similarity between the clients' data distributions in these majority classes.

Table 9: Community detection

Group ID	Members
$\boldsymbol{G_0}$	I_6
$\boldsymbol{G_1}$	I_8
$\boldsymbol{G_2}$	I_1
$\boldsymbol{G_3}$	I_5
$\boldsymbol{G_4}$	I_7
$\boldsymbol{G_5}$	I_9
$\boldsymbol{G_6}$	$I_2 + I_0 + I_3 + I_4$

4.2.7 Federated outlier detection

We compare as follows. We take two baselines as reference: (i) local, where clients only train on local data; and (ii) ideal, in which a client uses the model trained through federated learning on the set of clients sharing the same majority class (identified through the group identification phase). The test samples for each client are randomly sampled from the Fashion-MNIST test set, following the same inlier/outlier classes and the ratio of the corresponding client. The results presented in Table 10 show the average AUC-ROC scores across clients having the same inlier class. Our proposed configuration outperforms the local baseline, indicating that the federated outlier detection approach effectively leverages global information to improve the model's performance. Furthermore, our results are consistently close to the performance of the ideal baseline, demonstrating the potential of the methodology to achieve near-optimal results.

DRAFT

Table 10: AUC-ROC mean values summarising the performance of the proposed anomaly detection method

	Local	Our method	Ideal
Mean	0.649	0.728	0.740
Std dev	0.012	0.013	0.014

4.3 Efficient face-swapping using hardware-aware scaling

Preserving user privacy in DL applications, especially in the context of smart cities and surveillance applications, is fundamental. However, traditional methods for anonymising images, such as blurring and pixelation, can compromise downstream tasks and limit the effectiveness of the anonymised image. More advanced approaches, such as face-swapping algorithms, can remove personal information while preserving the image's quality and expression. Still, these methods are typically computationally intensive and cannot be run in real-time on resource-constrained devices.

Our work targets developing a many-to-many face-swapping technique that is lightweight and can be used for face anonymisation on edge devices with limited computational capabilities. The approach enables anonymisation pipelines compliant with the Privacy by Design principle.

4.3.1 Summary of the state-of-the-art

Advanced approaches such as face-swapping algorithms are effective ways to anonymise images but can be computationally intensive. Among the state-of-the-art approaches, two main techniques are used - inpainting-based approaches and GAN-based ones. Inpainting methods are computationally simpler and can replace the face area but may not preserve pose and expression. GAN-based approaches, instead, aim to generate high-fidelity images but at a higher computational cost. Among these, CIAGAN uses a discriminator network to generate different faces but requires retraining for each different target identity. SimSwap [5] and FaceShifter [6] allow for many-to-many identity mappings but come with even higher computational costs. Some works try to compress these techniques; however, they fail to reach complexity levels low enough for low-powered edge devices. Because of this, classical computer vision-based algorithms are typically used on embedded devices with minimal RAM.

4.3.2 Description of work performed so far

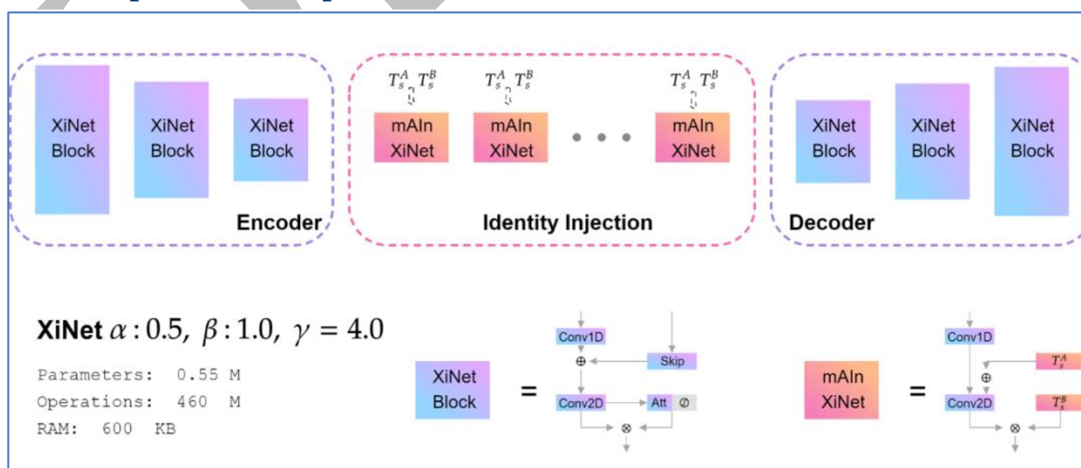

Figure 23. Block diagram of the proposed approach

DRAFT

We developed an approach based on SimSwap, managing to significantly compress the network (achieving a >90% reduction in operations and parameters) through two main steps:

- The original model has been replaced with a lightweight, easily scalable GAN model (XiNet)
- The generation of target face features has been moved from being computed at runtime to a pre-processing step, eliminating a significant portion of operations needed during the face-swapping operation

As of now, our approach has been adapted to run on a low-powered Risc-V MCU coupled with a tensor processing unit, managing speeds of over 10 frames per second.

4.3.3 Performance Evaluation

We evaluate the performance of our proposed approach comparing against SimSwap and FaceShifter considering the number of operations, the total number of parameters, and latency for what concern the computational complexity while we use the not reidentification rate and the pose MSE for the image quality, evaluating on the VggFace dataset.

Table 11: Comparative analysis of the performance of the proposed approach

Method	MAC	Parameters	Latency (Nvidia RTX2060)	Not reidentification rate	Pose MSE
Proposed	1266M	3.3 M	9.3 ms	98.8%	0.050
SimSwap	20621M	43.5 M	289 ms	98.1%	0.048
Faceshifter	97400M	350.0M	491 ms	99.4%	0.071

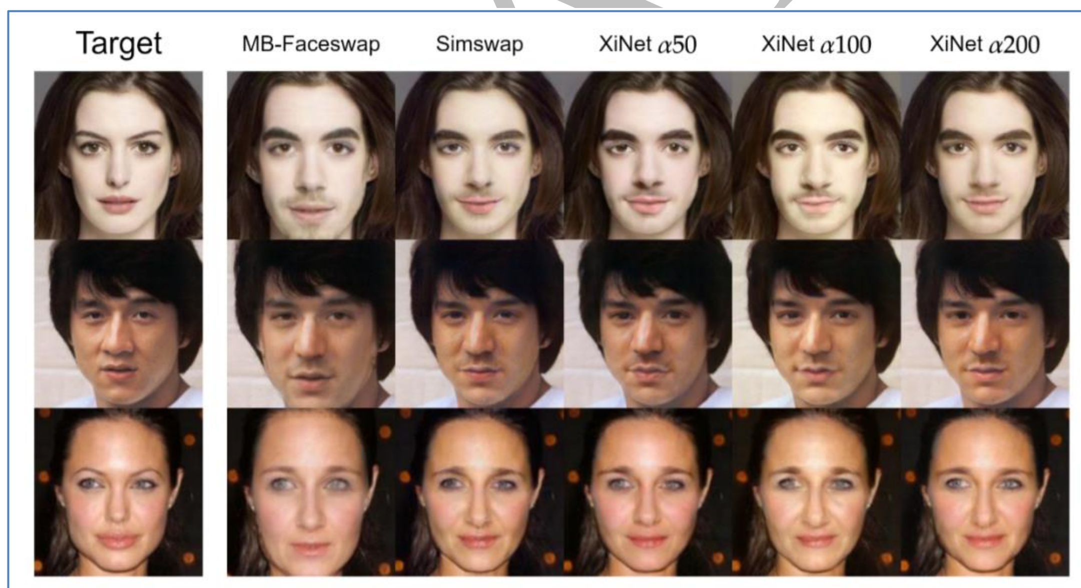

Figure 24. Qualitative evaluation of the proposed method against the two state-of-the-art approaches

5 KPIs

The section describes the relation of the main MARVEL components, associated with tasks T3.4 and T3.5, to the project KPIs and the corresponding component-related KPIs.

5.1 Project-related KPIs

KPI-O3-E2-1: *Model compression algorithms to achieve 70% compression rates, without noticeable degradation of accuracy.* Status: **Achieved**.

The results reported in the previous deliverable showed that, depending on the architecture to compress, it is possible to achieve different results. For MLP, we got 88% at no degradation on benchmark datasets. The compression scheme applied to AVCC and VCC ended up with 60% and 64% of compression with an accuracy degradation of less than 4%. Note that the final compression depends on a combination of model, and dataset target tasks.

KPI-O3-E2-2: *Optimise performance (prediction accuracy, time-to-decision) of DL deployment by 20%.* Status: **Achieved**.

This KPI is linked with the distributed execution of DL tasks. Towards that end, the implemented MARVEL E2F2C framework (Kubernetes cluster and MARVdash dashboard) enables DL task distributed execution, taking into consideration the efficient use of execution resources. MARVdash contributes to the ability to match the task resource requirements (GPU availability) to the various execution sites available in the MARVEL distributed environment. The choice of hardware greatly influences the time-to-decision. GPUs (Graphics Processing Units) can significantly speed up the computations involved in DL, resulting in faster decision times compared to using CPUs alone. Consequently, it is possible to enable improvements both in performance, particularly time-to-decision, as well as in the sophistication of the DL models being deployed, thus enhancing prediction accuracy. The optimisation goal of this KPI is achieved with the new deployment method offered by MARVdash (Section 3.2.1).

KPI-O3-E2-3: *Increase accuracy levels of real-time observations at the edge devices by 20%.* Status: **Achieved**.

This task is related to the deployment of the compressed models on edge devices, where the real-time requirements can be satisfied. Indirectly, it is possible to evaluate the FLOPs required to execute the model after compression and compare it with the FLOPs required by the original uncompressed model, by considering the compression of the model, as reported in **KPI-O3-E2-1.** We measured the inference time of the compressed models and compared it to the original one, achieving more than 4x speed-up. This would allow to increase by 4 times the number of images that can be processed in the same time window, by increasing the number of events that can be recognised.

iKPI-1.1: *At least three (3) tools for complex/federated/distributed systems handling extremely large volumes and streams of data*. Status: **Achieved**.

MARVdash contributes to this KPI indirectly, by enabling the instantiation of the FedL component (2.4.3). FedL is scalable to a large number of FL clients and capable of handling data from multiple sites arriving in a streaming fashion. Additional stream-handling tools that can be deployed through MARVdash are StreamHandler (2.3.1), DFB (2.3.2), DatAna (2.3.3), and HDD (2.3.4).

DRAFT

5.2 Asset Specific KPIs

- **MARVdash:** The asset-specific KPI related to MARVdash focuses on usability, specifically in terms of simplifying the process of specifying and automating component/service deployments. Additionally, user satisfaction with the MARVdash user interface is considered an important aspect of usability. The baseline for this KPI is the deployment of services in a Kubernetes cluster without utilising the user-facing front end of MARVdash. A second assessment round was conducted in February 2023 since the FORTH team believed that MARVdash users will be more familiar with the dashboard and their answers will be more indicative. The initial assessment involved a benchmarking process, as documented in D5.2[15]. According to the assessment results, the main functionalities of MARVdash were scored with an average ranging from 4.86 (lowest) to 6.79 (highest), which falls within the "Very good" category (with one rating in the "Excellent" category) in terms of qualitative assessment. The previous numbers were 4.75 and 6.22 correspondingly. Being more familiar with MARVdash, MARVEL partners rated the tool's functionalities with higher scores. Furthermore, the user experience assessment revealed that MARVdash performed above average compared to numerous other products. These results suggest that MARVdash has the potential to be successful in the market.
- **DynHP**: The component-related KPI focuses on providing interactive training. Since DynHP is a training methodology, it cannot be configured as a standalone service. The interaction with the user is required to configure the DNN training process, i.e., hyperparameter tuning, number of training epochs, etc. The DynHP component offers a Jupyter-lab environment through which it is possible to run and monitor the compression and training of a model on a specific dataset. Concerning the previous version, it is now compatible with Pytorch-lightning. This goes in the direction of focusing the effort only on the model implementation and compression configuration, leaving untouched the logic for compression.

DRAFT

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

6 Conclusion

In this document, we present the process of creating the infrastructure for deploying MARVEL components, which consists of a Kubernetes cluster. The cluster hosts MARVdash, a user-friendly dashboard service placed on top of the Kubernetes cluster. MARVdash facilitates interaction with the underlying E2F2C testbed by providing a landing page for users. Through MARVdash, users can launch services, design workflows, request resources, and specify execution parameters using a user-friendly interface.

Furthermore, we develop a deployment method that leverages MARVdash, allowing end users to select the execution environment for their applications without requiring a deep understanding of lower-level tools and interfaces.

To ensure efficient monitoring and management of the Kubernetes cluster, we install monitoring and managing tools such as Grafana, Prometheus, and Loki. These tools enable us to monitor and analyse the performance and health of the underlying infrastructure effectively.

Additionally, various aspects of model optimisation and efficient inference techniques were highlighted. The efficiency of inference processes was explored through the discussion of different methods and approaches. The utilisation of decentralised and unsupervised learning for achieving efficient anomaly detection was investigated. Furthermore, techniques for efficient face-swapping, utilising hardware-aware scaling to enhance performance, were also discussed. In summary, the focus was placed on the exploration of diverse approaches to optimise models, improve inference efficiency, enable efficient anomaly detection, and enhance face-swapping techniques through the implementation of hardware-aware strategies. Lastly, we described the contribution made to the MARVEL project and the KPIs related to the components involved.

In the future, optimization techniques are expected to undergo further improvements. Additionally, fine-tuning of components deployment will become more refined, allowing for better resource allocation and utilization, resulting in optimized workflows and reduced latency. These advancements will contribute to providing a stable infrastructure solution, ensuring reliable and consistent performance for various applications and services.

DRAFT

7 References

[1] Smart data models (2023) Smart Data Models. Available at: https://smartdatamodels.org/ (Accessed: 30 June 2023).

[2] T. P. Raptis and A. Passarella, "On Efficiently Partitioning a Topic in Apache Kafka," 2022 International Conference on Computer, Information and Telecommunication Systems (CITS), Piraeus, Greece, 2022, pp. 1-8, doi: 10.1109/CITS55221.2022.9832981.

[3] Kubernetes (2023) Kubernetes. Available at: https://kubernetes.io/ (Accessed: 30 June 2023).

[4] Song, Q., Wang, C., Wang, Y., Tai, Y., Wang, C., Li, J., Wu, J., & Ma, J. (2021). To Choose or to Fuse? Scale Selection for Crowd Counting. Proceedings of the AAAI Conference on Artificial Intelligence, 35(3), 2576-2583. https://doi.org/10.1609/aaai.v35i3.16360

[5] Chen, R., Chen, X., Ni, B., & Ge, Y. (2021). SimSwap: An Efficient Framework For High Fidelity Face Swapping

[6] Li, L., Bao, J., Yang, H., Chen, D., & Wen, F. (2019). FaceShifter: Towards High Fidelity And Occlusion Aware Face Swapping.

DRAFT