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

Big Data technologies and extreme-scale analytics

Multimodal Extreme Scale Data Analytics for Smart Cities Environments

D4.5: Security assurance and acceleration in E2F2C framework – final version[†]

Abstract: This deliverable is the final version of the security assurance and acceleration in the E2F2C framework and it describes the work conducted in Task 4.3 within WP4, after M18. In the context of MARVEL, security assurance is achieved using two components, namely EdgeSec Virtual Private Network (EdgeSec VPN) and EdgeSec Trusted Execution Environment (EdgeSec TEE). EdgeSec VPN secures the communication channels between the interconnected MARVEL components across the three layers (i.e., Edge, Fog, Cloud). EdgeSec TEE enables confidential and secure execution of applications that process sensitive data. Acceleration is accomplished with the GPURegex component that enables high-performance stream processing through GPU-accelerated pattern-matching. In this deliverable, we present the final versions of these tools, their relation to the MARVEL project, their role in the whole MARVEL framework, how they address the WP4 objectives, and how they achieve the KPIs defined.

Contractual Date of Delivery	30/06/2023
Actual Date of Delivery	30/06/2023
Deliverable Security Class	Public
Editors	*Eva Papadogiannaki, Anthi Barmpaki, Emmanouil Michalodimitrakis (FORTH)*
Contributors	FORTH, UNS, FBK, MT, ATOS, ZELUS
Quality Assurance	*Borja Saez Mingorance (IFAG)* *Nikos Nikolaou (ITML)*

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

The *MARVEL* Consortium

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

Document Revisions & Quality Assurance

Internal Reviewers

1. *Borja Saez Mingorance, IFAG*
2. *Nikos Nikolaou, ITML*

Revisions

Version	Date	By	Overview
6.0.0	30/06/2023	FORTH	Final quality check and approval
5.0.0	28/06/2023	Eva Papadogiannaki (FORTH)	Final document
4.0.0	27/06/2023	Eva Papadogiannaki (FORTH)	Incorporated reviews Document ready for IR (round 2)
3.2.0	26/06/2023	Borja Saez Mingorance (IFAG)	IR (round 1)
3.1.0	26/06/2023	Nikos Nikolaou (ITML)	IR (round 1)
3.0.0	21/06/2023	Eva Papadogiannaki (FORTH)	Document ready for IR
2.0.0	19/06/2023	Eva Papadogiannaki, Anthi Barmpaki, Emmanouil Michalodimitrakis (FORTH)	Document partially ready for IR
1.0.0	29/05/2023	Eva Papadogiannaki (FORTH)	Incorporated input from contributors
0.2.0	10/05/2023	Eva Papadogiannaki (FORTH)	Comments addressed - Final ToC
0.1.2	02/05/2023	Dragana Bajovic (UNS)	Comments on the ToC
0.1.1	13/04/2023	Borja Saez Mingorance (IFAG)	Comments on the ToC
0.1.0	06/04/2023	Eva Papadogiannaki (FORTH)	ToC

Disclaimer

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

Table of Contents

List of Tables

List of Figures

List of Abbreviations

AAC	Automated Audio Captioning
AS	Authenticator Server
API	Application Programming Interface
CPU	Central Processing Unit
DFA	Deterministic Finite Automaton
DFB	Data Fusion Bus
DoA	Description of Action
DRAM	Dynamic Random Access Memory
EC	European Commission
EPC	Enclave Page Cache
IDS	Intrusion Detection System
IETF	Internet Engineering Task Force
IP	Internet Protocol
IPSec	Internet Protocol Security
ISA	Instruction Set Architecture
ISP	Internet Service Provider
GPGPU	General Purpose Graphics Processing Unit
GPU	Graphics Processing Unit
GRE	Generic Routing Encapsulation
KPI	Key Performance Indicator
L2TP	Layer 2 Tunnelling Protocol
MEE	Memory Encryption Engine
OS	Operating System
PD	Personal Data
PPTP	Point-to-Point Tunnelling Protocol
PRM	Processor Reserved Memory
R2	2nd Release of the MARVEL Integrated framework
RNA	Ribonucleic Acid
RTT	Round-Trip Time
SGX	Software Guard Extensions
SIMD	Single Instruction Multiple Data
SOCKS	Socket Secure
SSL	Secure Sockets Layer
TCB	Trusted Computing Base
TEE	Trusted Execution Environment
TLS	Transport Layer Security
VM	Virtual Machine
VPN	Virtual Private Network
WP	Work Package

Executive Summary

The goal of this deliverable is to present and describe the final versions of the components that participate in the whole MARVEL platform and offer security and acceleration features in the complete Edge-to-Fog-to-Cloud continuum. This deliverable has been developed within the scope of WP4 “MARVEL E2F2C distributed ubiquitous computing framework”, and more specifically, within the scope of Task 4.3 “Security and acceleration in the complete E2F2C” of the MARVEL project under Grant Agreement No. 957337.

The initial versions of the security and acceleration components were presented in deliverable D4.2 on M18, while this deliverable demonstrates their final versions that have been developed in Task 4.3, during the period M19 – M30.

EdgeSec Virtual Private Network (EdgeSec VPN) and EdgeSec Trusted Execution Environment (EdgeSec TEE) are the two components that offer security. EdgeSec VPN secures the communications between the interconnected components of the MARVEL platform within the three distinct layers (i.e., Edge, Fog, Cloud). EdgeSec Trusted Execution Environment (EdgeSec TEE), enables trusted and secure execution of applications that process sensitive data. GPURegex is a component that offers GPU-accelerated stream processing and more specifically, it accelerates the pattern-matching procedure. In this deliverable, we discuss the final versions of the three components, their relation to the MARVEL project, how they address the WP4 objectives, and how they achieve the KPIs defined.

1 Introduction

1.1 Purpose and Scope

This deliverable, entitled D4.5 "Security assurance and acceleration in the E2F2C framework – final version" presents the work performed in the context of Task 4.3 "Security and acceleration in the complete E2F2C", within the scope of WP4 "MARVEL E2F2C distributed ubiquitous computing framework" and the MARVEL project. As this is the final version of the deliverable, we present the conclusive versions of the components that have been developed in the context of the MARVEL project and Task 4.3 after M18.

The components that are demonstrated in this deliverable and offer security features to the E2F2C framework are (i) EdgeSec VPN and (ii) EdgeSec TEE. EdgeSec VPN secures the data transfer over the network, while EdgeSec TEE offers confidential computing for Python applications that process sensitive user data. In the context of the MARVEL project, EdgeSec VPN encrypts any data transferred between the MARVEL components to meet the requirements of strict communication security. EdgeSec TEE offers trusted execution for sensitive data.

The component that offers acceleration in the E2F2C framework is GPURegex. More specifically, GPURegex enables the processing performance acceleration of the pattern-matching procedure. In the context of the MARVEL project, GPURegex facilitates fast keyword searching against captions exported from audio and video captures. Once a keyword is matched, we receive an indication that a specific event is detected.

In this deliverable, we discuss the details of the development, deployment, integration, and performance results of the three components that bring security assurance (i.e., EdgeSec VPN, EdgeSec TEE) and acceleration (i.e., GPURegex) in the MARVEL E2F2C framework. Since this is the second and final version of the deliverable that presents the work conducted in Task 4.3 "Security and acceleration in the complete E2F2C" after M18, we choose not to repeat information that can be also found in D4.2[1], to avoid text reduplications. Thus, since in this final version of the deliverable we elaborate only on the final component versions and modifications (when applied), we encourage the reader to refer to the D4.2 as well, for further details.

1.2 Relation to other Work Packages, Deliverables, and Activities

The main features that are primarily expected from WP4 and Task 4.3 are (i) secure data transfers, with data that originate from sensors and arrive at the relevant components across the different layers of MARVEL aiming to be processed in a trusted manner, and (ii) acceleration during the processing of streaming data. In the paragraph below, we include the description of Task 4.3, as retrieved from the Description of Action (DoA).

> *This task will explore remote attestation, a well-known technique, for verifying the state of remote computing devices and for verifying the trustworthiness of the data collected and shared by remote sensors. Moreover, trusted execution environments (such as Intel SGX) which can enable remote attestation and further provide full memory encryption will be explored as well (Sect. 1.4.1.5). These environments contain secure elements that lie in the hardware chip, and support, at least, advanced cryptographic functions and physically protected storage of private and secret keys. This will allow building* ***a multi-***

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

> ***layer architecture*** *that will provide* ***security, trust, and privacy*** *in the* ***edge device*** *itself. Finally, the utilisation of GPU-accelerated* ***streaming processing*** *in edge devices will also be explored in this task (Sect. 1.4.1.5). This low-end GPU acceleration in the processing of streaming data can accelerate light computations as a pre-processing phase, right* ***before offloading tasks to the cloud****.*

Specifically, EdgeSec VPN participates in the MARVEL platform as a holistic element, securing every step of data transmission with end-to-end network packet encryption. Hence, this deliverable is significantly related with Task 3.4, where EdgeSec VPN participates in the distribution of the AI tasks. Also, it interacts with Task 3.3, since GPURegex accelerates the processing of audio/video captioning data resulting from components that participate there. The three components are also related to WP5 and WP6, since they participate in the integration process, the pilot realisation, and final evaluation. Finally, Task 4.3 and the related deliverables (i.e., D4.2 and D4.5) have also close relation to WP1, since they address the project objectives and the respecting KPIs.

As already presented in the initial version of this deliverable (D4.2), Figure 1 displays the conceptual architecture of the whole MARVEL platform, where the placement of the components EdgeSec VPN, EdgeSec TEE, and GPURegex is highlighted using black, dashed rectangles. Particularly, EdgeSec VPN and EdgeSec TEE participate in the Security, Privacy, and Data Protection subsystem, whereas GPURegex is operating in the Optimised E2F2C Processing and Deployment subsystem.

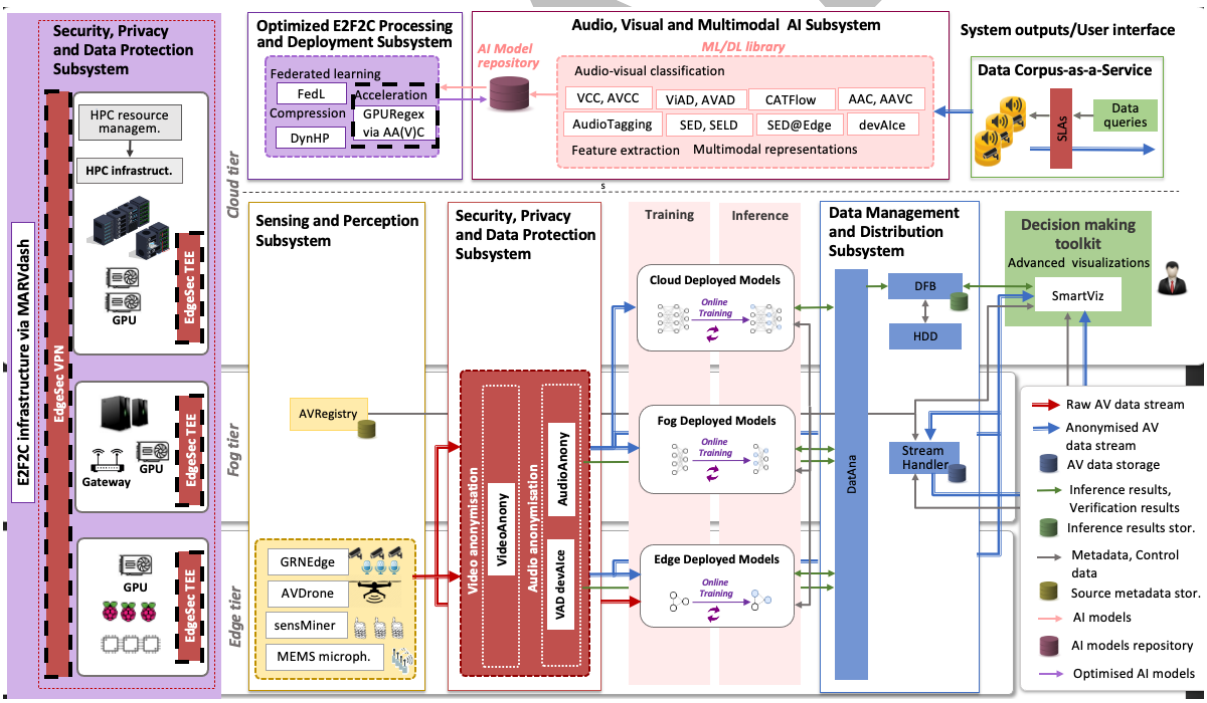

Figure 1. Conceptual architecture of the whole MARVEL platform

1.3 Contribution to WP4 and Project Objectives

This deliverable is the main and final outcome of the work performed in Task 4.3, which is entitled "Security and Acceleration in the complete E2F2C". Task 4.3 is part of WP4 "MARVEL E2F2C distributed ubiquitous computing framework". The description of WP4, which can be also found in the DoA, is presented in the following paragraph:

> *WP4 develops the MARVEL E2F2C framework to fully harness, in a **resource-optimised** and **secure way**, the edge (including data capturing), fog and cloud resources to effectively orchestrate and distribute computational and AI-related tasks (Pillar III). A major consideration towards this goal is the ability to perform a significant part of the processing at the Edge or Fog. Therefore, AI-enabled devices that can run (edge-optimised) light-weight DL models play a key role. Moreover, **security** and **acceleration** methods will be enforced at all layers so as to end up with a robust, trustworthy, and fast decision support toolkit.*

The objectives of WP4, which can be also found in the DoA, are presented in this paragraph:

> *To address these challenges, WP4 objectives are to: **(i) offer GPU stream processing acceleration in edge devices and nodes; (ii) define and deploy a security strategy, including privacy-aware algorithms, at the edge;** (iii) describe innovations performed by analogue and digital microphones that are based on MEMS technology; (iv) utilise the openSMILE platform for audio analysis and feature extraction; (v) develop advanced visualisation techniques to support both real-time and long term decision-making.*

As expected, this deliverable contributes to WP4 and its project objectives by offering: (i) stream processing acceleration with the GPURegex component and (ii) a security strategy that combines EdgeSec VPN and EdgeSec TEE, the two MARVEL components that propose the utilisation of privacy-aware technologies (i.e., encrypted data transfer across the network and Trusted Execution Environments for confidential execution) for the establishment of security and privacy across MARVEL components, at the edge and other layers of operation.

1.4 Structure of the Report

Similar to the initial version of this deliverable (D4.2), the structure of the final version of the deliverable is outlined as follows:

- In this section, we provide an introduction of this deliverable, highlighting its purpose and scope, its relation to the other work packages, deliverables, and activities within the context of the MARVEL project, its contribution to WP4, and the total objectives of the project (i.e., Section 1).
- In the second section, we present EdgeSec VPN (i.e., Section 2), which offers encryption of data transmitted over the network traffic.
- Then, in the third section, we present EdgeSec TEE (i.e., Section 3), which offers security and privacy in the complete E2F2C2, by enabling the confidential processing of code and sensitive data into isolated and encrypted memory regions.
- In the fourth section, we present details about the component, namely GPURegex, that offers processing acceleration of pattern-matching applications, using GPUs or other hardware devices that enable SIMD processing (i.e., Section 4).
- In the final section, we conclude this deliverable, by summarising its contents and results (i.e., Section 5).

2 Secure Communication on the Edge (EdgeSec VPN)

2.1 Introduction and Objectives

EdgeSec VPN is the first component that brings security and privacy features to the complete E2F2C framework, within the MARVEL project. More specifically, EdgeSec VPN is the n2n product of ntop which is based on the technology of peer-to-peer VPNs and is configured to satisfy the needs of MARVEL.

In the following sections, we provide some background information regarding VPNs, secure peer-to-peer communications, and a summary of the state-of-the-art. We describe the basic development and deployment details. Then, we locate the relevant project and component KPIs. Finally, we discuss about their correlation with EdgeSec VPN and how they can be realised within the context of MARVEL.

2.1.1 Contributions

EdgeSec is a suite of components and the component that is referred to as EdgeSec-VPN is the software n2n[2] from ntop. n2n is a lightweight VPN, designed to create virtual networks that bypass intermediate firewalls with ease.

EdgeSec VPN can offer several benefits to MARVEL E2F2C cluster architecture. Firstly, EdgeSec VPN enables the establishment of an E2F2C Kubernetes cluster. Also, EdgeSec VPN offers secure communication channels between the different components of the cluster, including edge devices, fog nodes, and cloud infrastructure. All data transmitted between these components is encrypted, ensuring confidentiality and integrity. Additionally, EdgeSec VPN offers data privacy by masking the actual IP addresses and network information of the cluster's components. This helps protect sensitive information and prevents unauthorised access or interception of data. With EdgeSec VPN, MARVEL can enforce access control policies to restrict connectivity and communication to authorised devices or networks.

Overall, EdgeSec VPN enhances the security, privacy, and flexibility of the MARVEL E2F2C cluster, enabling secure and efficient communication between the different components of the cluster while safeguarding data and network integrity.

2.1.2 Summary of the State-of-the-Art

As already presented in D4.2, among the most common VPN solutions are OpenVPN [1], Wireguard [2], and L2TP/IPSec [2][4]. All of the above are fully functional in Windows, macOS as well as Linux operating systems.

OpenVPN utilises SSL/TLS for key exchange and encryption. It is open-source and has been recognised as having no major vulnerabilities offering a range of ciphers and encryption methods.

L2TP/IPSec is a combination of the L2TP protocol along with the IPSec to provide encryption and key negotiation. IPSec can operate in transport mode or Tunnel mode. In transport mode, IPSec encrypts the data payload of the IP packet while leaving the header as is. On the other hand, in tunnel mode, IPSec encapsulates the entire IP packet within a new packet meaning that the payload as well as the original header are encrypted.

WireGuard utilises asymmetric key cryptography and state-of-the-art cryptographic algorithms and protocols. WireGuard claims to be faster, simpler, and leaner than other VPN solutions.

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

In summary, L2TP/IPSec preceded OpenVPN and the latter is considered nowadays to be the first choice. WireGuard is a more recent solution that aims to be simpler and faster.

2.1.3 Component Modifications after D4.2

Since D4.2, no modifications were made to EdgeSec VPN mainly due to the reason that is based on a ready solution which is the n2n. The decision was made to ensure the stability and reliability of the component by maintaining the integrity of the original design and minimise potential risks that may be introduced by any modifications. During R2 (2nd Release of the MARVEL Framework) the effort was focused on integrating the EdgeSec VPN with all the nodes that participate in the Kubernetes cluster as well as a few other nodes, reaching a total of 18 participating computing devices (Table 1). The integration process involved meticulous planning and implementation to ensure seamless communication and secure connections among the nodes. Through this effort, the network infrastructure was expanded, allowing for increased connectivity across the nodes.

2.2 Deployment and Integration

EdgeSec VPN as stated before is the open-source software n2n. The latest version of EdgeSec VPN is uploaded on the MARVEL Docker image registry and can be downloaded from the MARVEL platform. In this section, we will describe the current development, deployment, and integration status of the component, namely EdgeSec VPN, concerning the whole MARVEL platform.

2.2.1 Development

As no modifications were made since D4.2, the development of EdgeSec VPN remained the same as presented in D4.2. Three VMs were used, one Super Node, and two edge nodes. The operating system of the VMs is Ubuntu Linux version 18.04 LTS, the size of the RAM is 4GB, and the disk is 40GB.

The component is containerised with Docker for easier deployment. For EdgeSec VPN, the first layer of the Docker image is based on the Ubuntu:18.04 image. The resulting EdgeSec VPN image can be found on the MARVEL Docker image registry and can be downloaded or shared upon request.

2.2.2 Deployment

The architecture of the EdgeSec VPN is identical to the architecture of n2n as already presented in D4.2. For completeness, we present Figure 2 again.

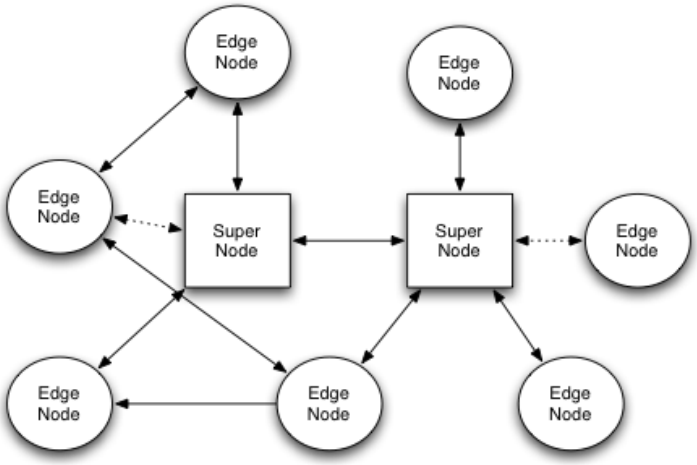

Figure 2. The n2n architecture

n2n and consecutively EdgeSec VPN, consists of two crucial components: edge nodes and Super Nodes. The edge nodes represent the participating peers within the network. On the other hand, Super Nodes enable the edge nodes to discover other edge nodes. They also handle traffic routing when nodes are situated behind symmetrical firewalls.

Operated at the second layer of the OSI model, the n2n and EdgeSec VPN allow peers to effectively traverse NAT and firewalls, ensuring their reachability. Edge nodes belonging to the same virtual network come together as a community. While Super Nodes can serve multiple communities, a single computer can join multiple communities.

Within a community, packet encryption is achievable through the utilisation of an encryption key. Edge nodes establish direct communication with each other using UDP. However, in cases where direct communication is not possible due to specific NAT circumstances, the Super Node can facilitate packet relay.

The Docker file used has been presented in D4.2 and remains the same (Listing 1.)

```
FROM ubuntu:18.04
RUN apt-get update && apt-get install -y build-essential net-tools autoconf pkg-
config
RUN mkdir -p /usr/ipsec
WORKDIR /usr/ipsec
COPY ./ .
RUN ./autogen.sh
RUN ./configure
RUN make
RUN make install
EXPOSE 4194/udp
CMD ["sh", "init_script.sh"]
```

Listing 1: EdgeSec VPN Dockerfile

After the initial build of the image, the Super Node can be started in a machine that has port 4194 exposed to the Internet. Then, we are ready to connect the respecting edge nodes.

2.2.3 Integration with MARVEL

To deploy the components in MARVEL, a multi-layered architecture involving the edge, fog, and cloud layers is necessary. To effectively manage the deployment using Kubernetes, all three layers need to be included within the Kubernetes cluster. However, Kubernetes inherently relies on pod-to-pod communication without the presence of NAT, which poses a challenge when dealing with remote nodes.

The EdgeSec VPN offers a solution in this scenario by seamlessly connecting all participating nodes as if they were part of the same local network. This capability ensures that any NAT or firewall restrictions become transparent, enabling seamless communication between the nodes. Thus, EdgeSec VPN addresses the conflict between Kubernetes' communication requirements and the actual setup involving remote nodes, facilitating effective deployment management.

A full list of participating nodes with their description is presented in Table 1.

Table 1: EdgeSec VPN participating nodes

E2F2C Layer	Nodes	Description
Cloud	Super Node	VM used by the edge nodes for discovering other edge nodes
	Master	VM used in Kubernetes cluster with the role of master in the cloud
	worker1	VM used in Kubernetes cluster with the role of worker in the cloud
	worker2	VM used in Kubernetes cluster with the role of worker in the cloud
	worker3	VM used in Kubernetes cluster with the role of worker in the cloud
	worker4	VM used in Kubernetes cluster with the role of worker in the cloud
	GPU worker	Physical Machine Server with GPU used in Kubernetes cluster with the role of worker in the cloud
	Zabbix server	VM used for collecting resource metrics from all involved nodes in all layers
Fog	GRN fog 2	Physical Machine Server with GPU used in Kubernetes cluster with the role of worker at the fog layer
	UNS fog 1	Physical Machine Server with GPU used in Kubernetes cluster with the role of worker at the fog layer
	MT fog 2	Physical Machine Server with GPU used in Kubernetes cluster with the role of worker at the fog layer
Edge	GRN edge 1	Physical Machine Desktop used in Kubernetes cluster with the role of worker at the edge layer
	GRN edge 2	Physical Machine Desktop used in Kubernetes cluster with the role of worker at the edge layer
	GRN edge 3	Jetson used in Kubernetes cluster with the role of worker at the edge layer
	GRN edge 4	Jetson used in Kubernetes cluster with the role of worker at the edge layer
	UNS edge 1	Intel NUC used in Kubernetes cluster with the role of worker at the edge layer
	UNS edge 2	Raspberry Pi used in Kubernetes cluster with the role of worker at the edge layer
	UNS edge 3	Physical Machine Laptop used in Kubernetes cluster with the role of worker at the edge layer

Essentially, all participating computing devices form a full mesh network where every device has a direct connection with every other device, as depicted in Figure 3.

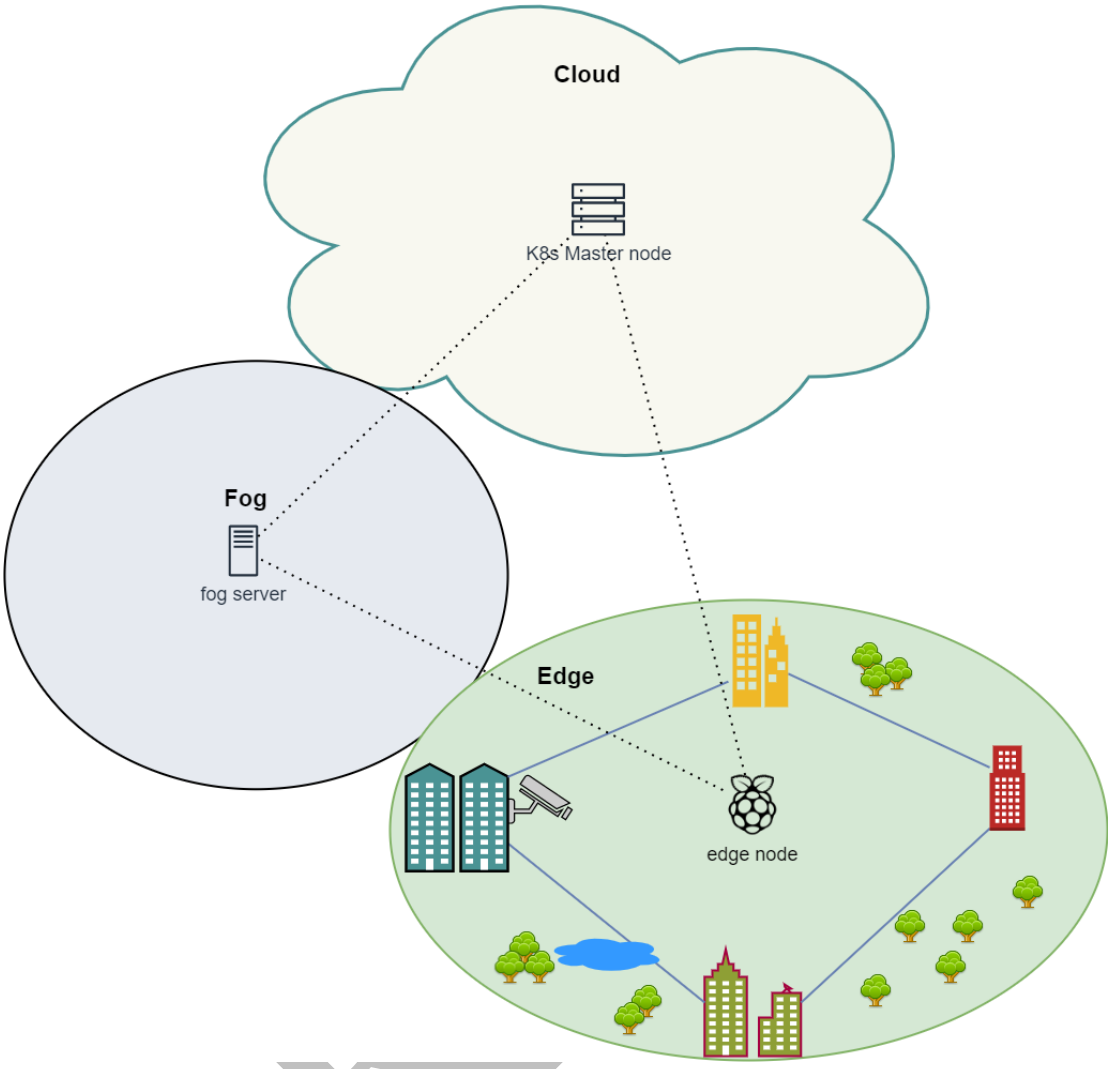

Figure 3. EdgeSec VPN mesh topology

2.3 Use Cases and Related Components

This section describes the MARVEL use cases in which EdgeSec VPN participates. In addition, an overview of the related components is presented. EdgeSec VPN secures the communications between every component that either participates in the edge, fog, or cloud layer of MARVEL.

2.3.1 Related Components

As explained in D4.2, the majority of MARVEL components are deployed within Kubernetes. However, not all components deployed in Kubernetes are intended to operate solely in the cloud layer.

In the context of Kubernetes, seamless pod-to-pod communication across nodes is essential. This comes in direct contradiction with the actual setup of having remote nodes. The adoption of EdgeSec VPN overcomes this problem, because it brings together all the participating nodes as if they were under the same local network making any NAT or firewall transparent to the communication between them.

EdgeSec VPN serves as the backbone network that allows each remote node to join the Kubernetes cluster in the cloud. In essence, all components that are deployed in Kubernetes utilise the EdgeSec VPN as they traverse the tunnel created by it.

2.3.2 EdgeSec VPN in UNS Use Cases

Initially, for the UNS use cases, we had to deal with the fact that all the involved computing devices are accessing the internet via proxy. This required modifying the Dockerfile to include required proxy variables as well as the creation of an additional Super Node. However, after D4.2 was released, the UNS infrastructure was modified and no proxy was used. This made the proxy variables as well the additional Super Node obsolete.

The infrastructure of UNS consisted of a server that is located at the fog layer, a Raspberry Pi that is located at the edge layer, and an Intel NUC that is mounted on a drone also located at the edge. In addition to those and after the release of D4.2, a new device was added to the edge layer and more specifically a laptop. The resources of all involved devices are monitored via a Zabbix server which is located in the cloud. The monitoring of the devices at the edge and the fog is feasible due to the presence of EdgeSec VPN. All the participating EdgeSec VPN nodes for the UNS cases are presented in Figure 4.

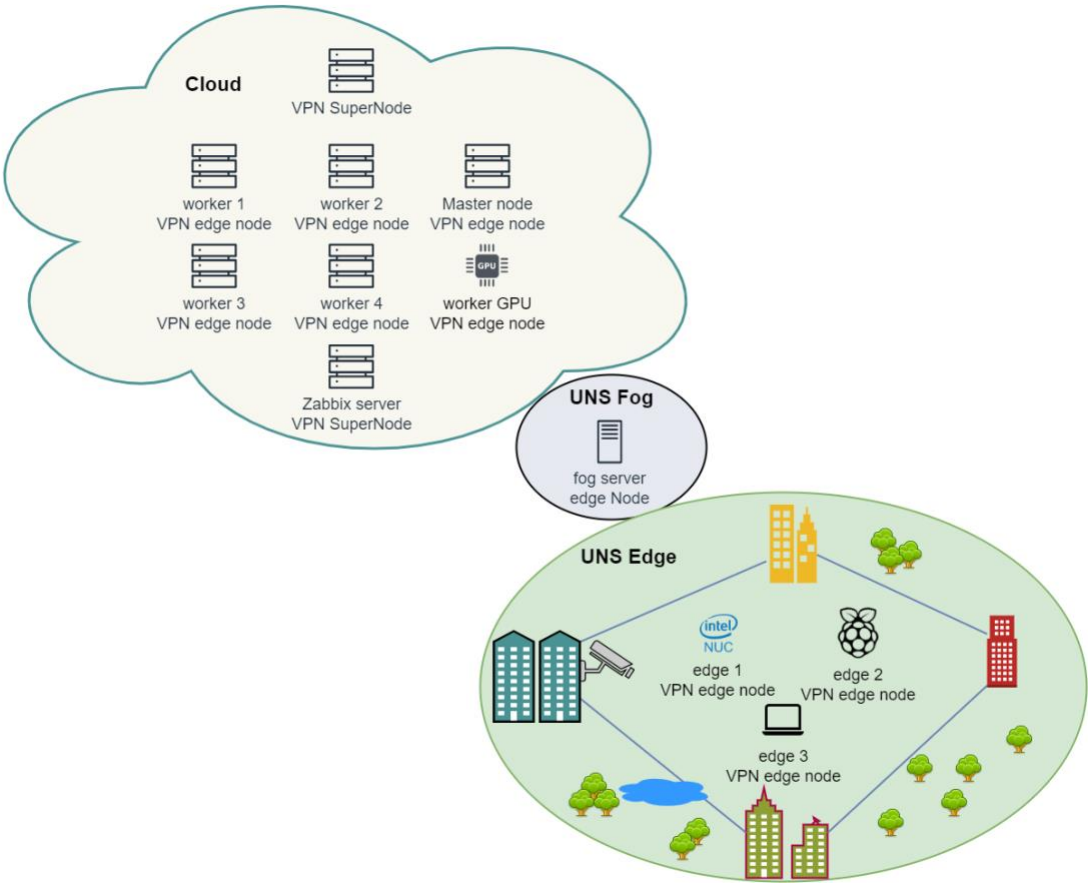

Figure 4. EdgeSec VPN in UNS Pilot

2.3.3 EdgeSec VPN in GRN Use Cases

For the GRN use cases, no proxy was present in the infrastructure, and therefore no extra modification was required to the initial Dockerfile. The infrastructure of GRN consisted of a server that is located at the fog layer and a workstation located at the edge layer. In addition to those and after the release of D4.2, new devices were added to the edge layer and more specifically a secondary workstation as well as two NVIDIA Jetson devices.

The communication between those nodes at the GRN and the Super Node in the cloud in PSNC's infrastructure is unhindered and therefore no additional Super Node was required. Nodes are able to directly announce themselves and discover other nodes via the Super Node.

The resources of all involved devices are monitored via a Zabbix server which is located in the cloud. The monitoring of the devices at the edge and at the fog is feasible due to the presence of EdgeSec VPN. All the participating EdgeSec VPN nodes for the GRN cases are presented in Figure 5.

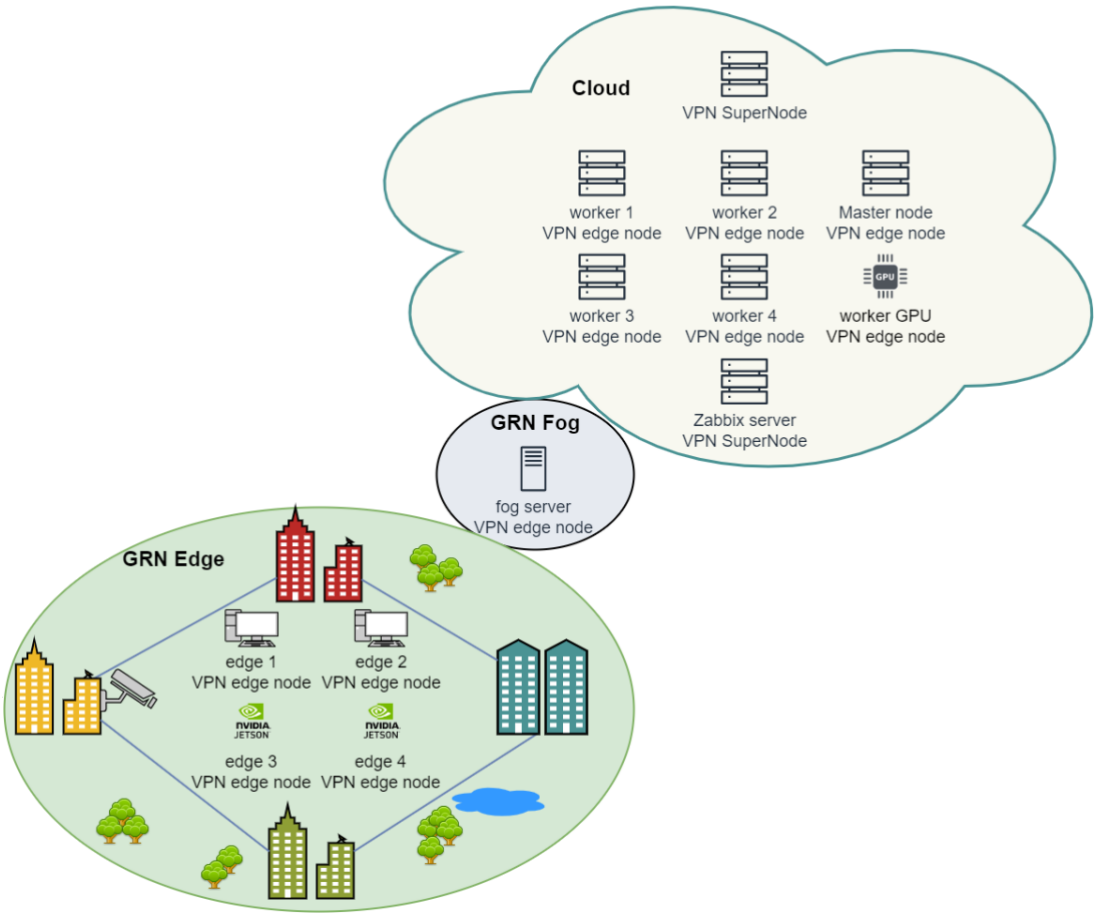

Figure 5. EdgeSec VPN in GRN Pilot

2.3.4 EdgeSec VPN in MT Use Cases

For the MT use cases no changes were made after the release of D4.2. The fog layer of the infrastructure for the MT use cases is actually hosted in FBK's infrastructure. FBK is hosting two workstations that are located at the fog layer. Only one of them is meant to be part of the EdgeSec VPN. The communication between the workstation at the FBK and the Super Node in the cloud in PSNC's infrastructure is unhindered and therefore no additional Super Node was

required. The workstation is able to directly announce itself and discover other nodes via the Super Node.

The resources of all involved devices are monitored via a Zabbix server which is located in the cloud. The monitoring of the server at the fog is feasible due to the presence of EdgeSec VPN. All the participating EdgeSec VPN nodes for the MT cases are presented in Figure 6.

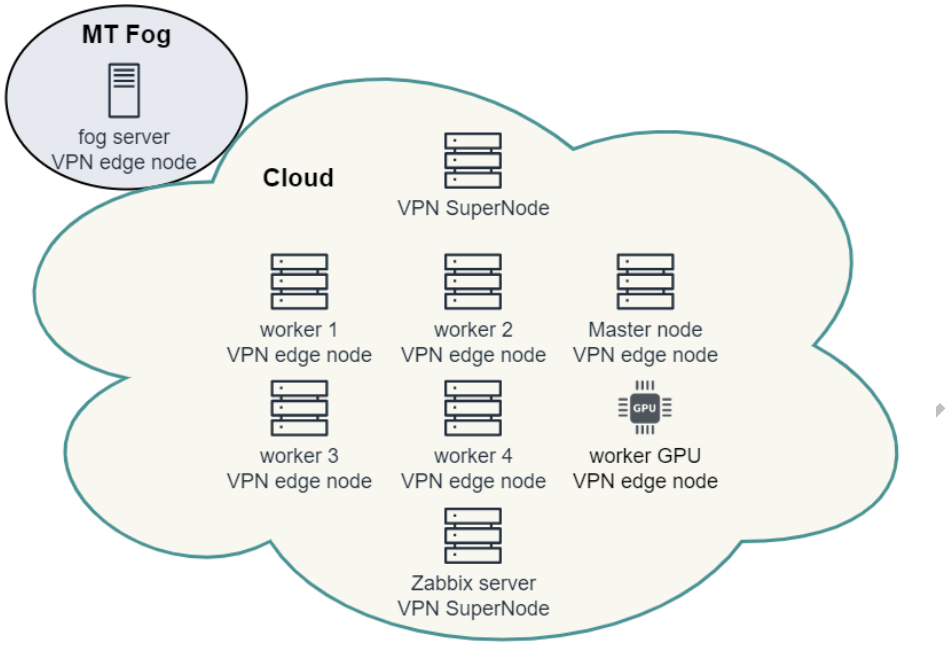

Figure 6. EdgeSec VPN in MT Pilot

2.4 Experimental Results

This section demonstrates the experiments that were performed to explore the capabilities of EdgeSec VPN.

2.4.1 Testbed Setup

The testbed that was used before the release of D4.2 was consisted of three VMs that were created using Proxmox as hypervisor. In this version the testbed that is used is the actual infrastructure used by MARVEL and presented in Table 1.

2.4.2 Experiments

Following the methodology that was presented in D4.2, we repeat the experiments this time on the actual MARVEL infrastructure. Pings via the local interface as well as the VPN interfaces are conducted to test the connectivity with and without the deployment of EdgeSec VPN. We send 100 ping requests from the Kubernetes master node to the Kubernetes worker1 node using the local IPs and then we do the same using the VPN IPs (Figure 7). In both cases, we have 0% packet loss.

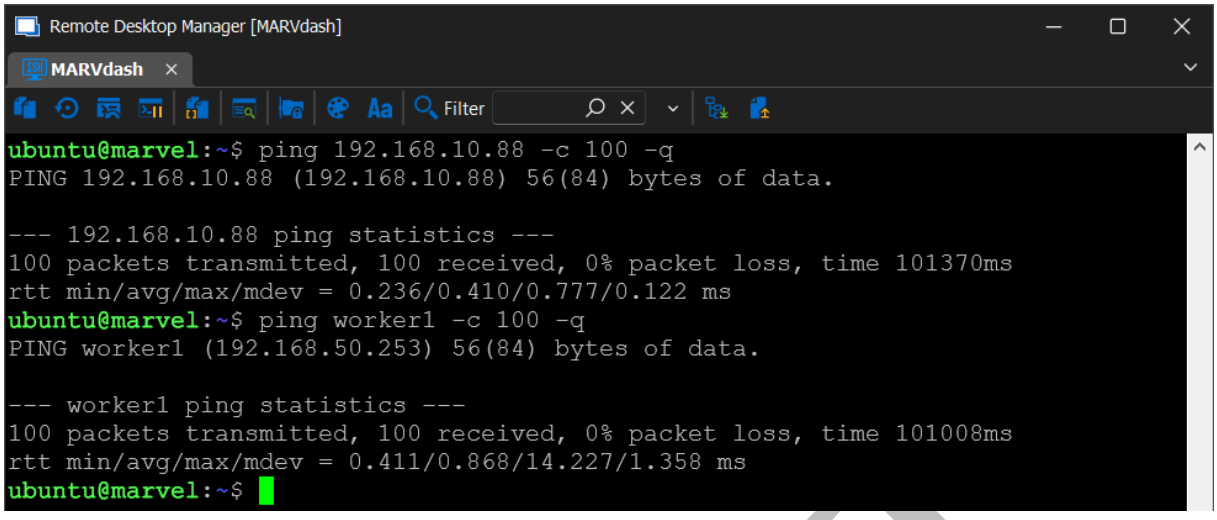

Figure 7. Pings from the Kubernetes master node to Kubernetes worker1 node, using local and VPN IP

Furthermore, to verify that data in transit is encrypted, we run a simple Python-based http server on the worker1 node (Figure 8) and we attempt to request the contents of the webpage from the master node using both the local as well as the VPN IP. In the left side of Figure 8, we can see that in both cases the same http content is returned. On the right side of the same figure, we see the requests made to the http server on the local and the VPN IP. The verification of the encryption is showcased in Figure 18 of Section 2.4.3.

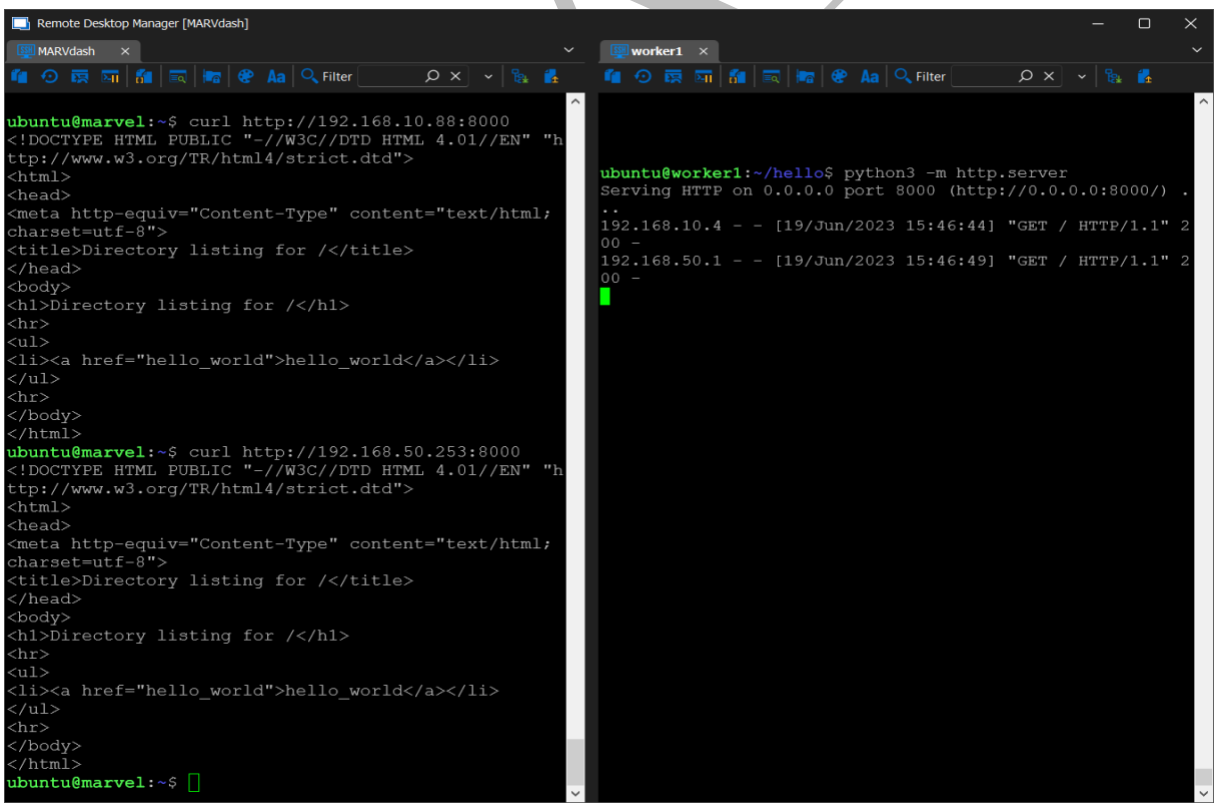

Figure 8. Request to the http server using the local and VPN IP

2.4.3 Final Results

In order to test the overhead that is introduced by the deployment of EdgeSec VPN (in terms of network traffic latency), we use the ping command again on the actual infrastructure. First, we send 20 ping requests from the first edge node to the second edge node using the public Ips and then we do the same using the VPN Ips. We repeat the same test with payload 100 bytes, 500 bytes, and 1000 bytes (worker1 is displayed in Figure 9 and Figure 10, worker2 is displayed in Figure 11 and Figure 12, worker3 is displayed in Figure 13 and Figure 14, worker4 is displayed in Figure 15 and Figure 16). The ICMP header and IP version 4 header add extra 28 bytes to the packet.

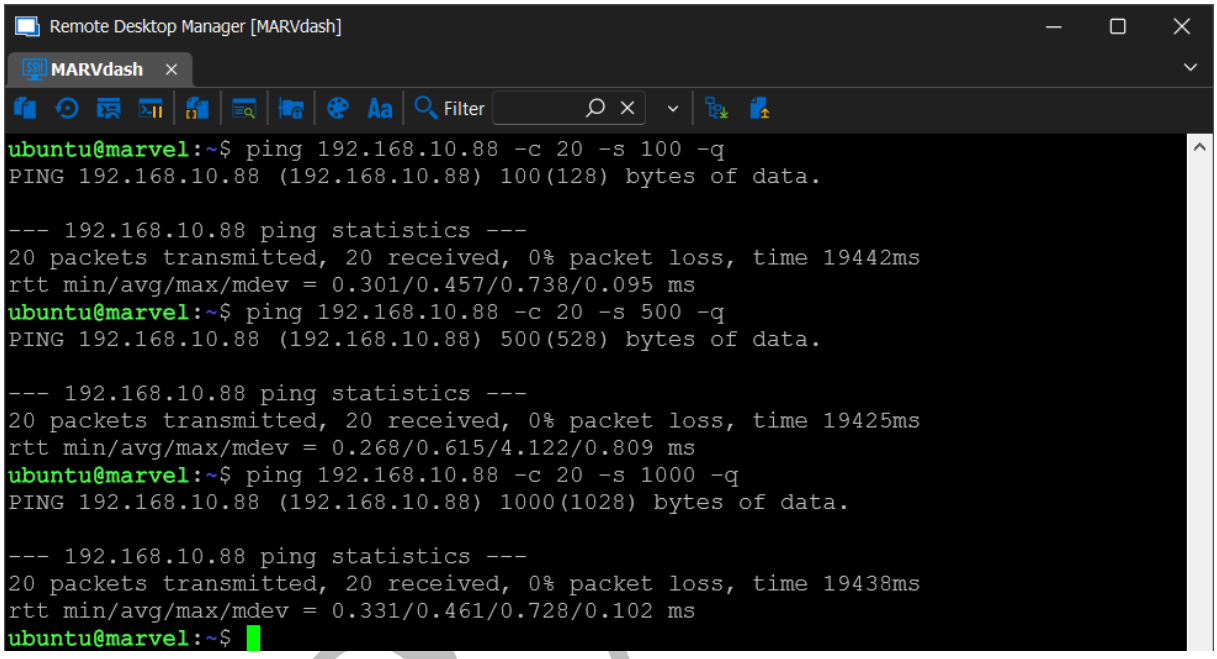

Figure 9. Master node pings the local IP of worker1 with payload 100 bytes, 500 bytes and 1000 bytes

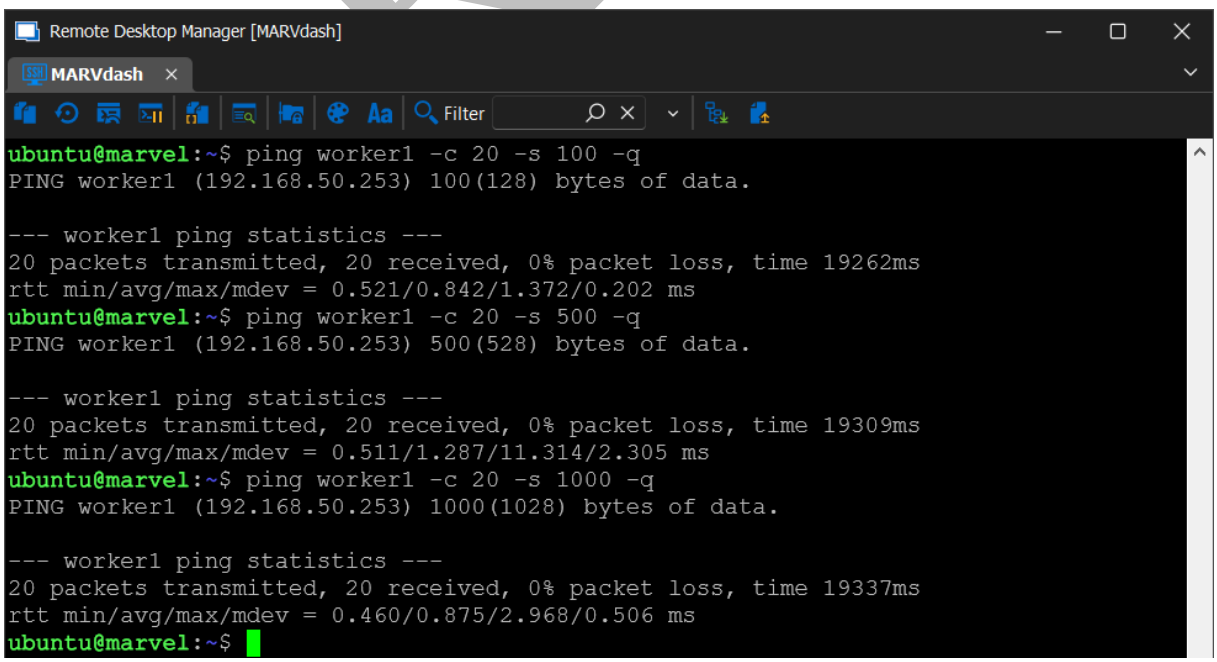

Figure 10. Master node pings the VPN IP of worker1 with payload 100 bytes, 500 bytes and 1000 bytes

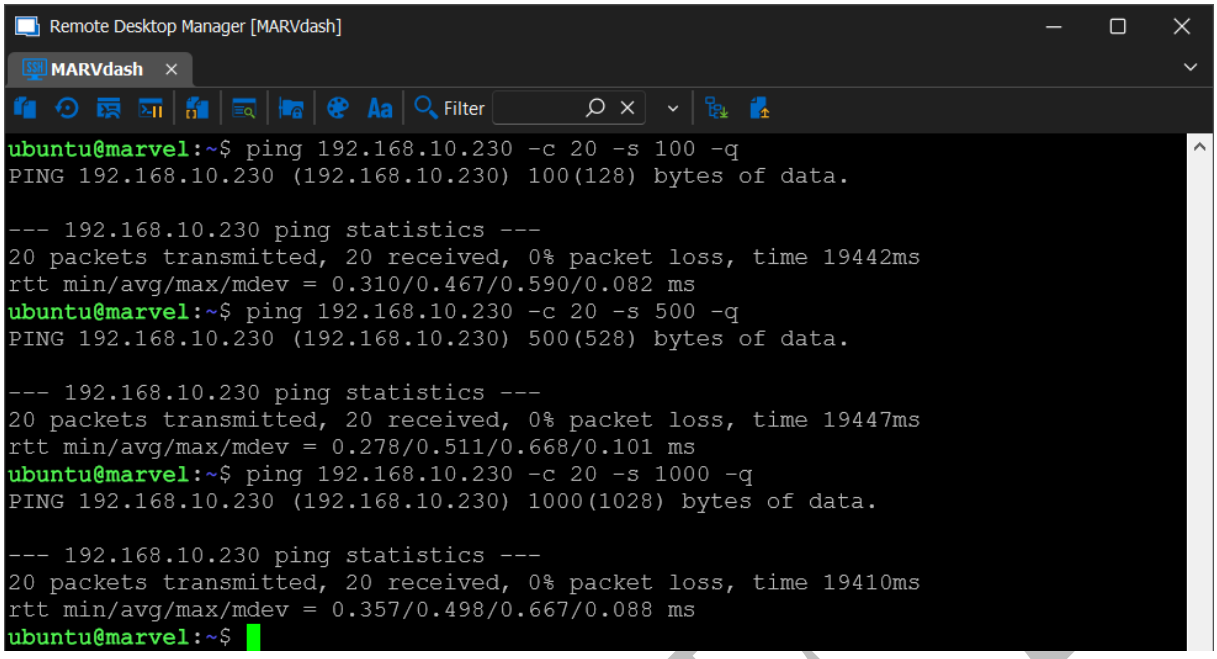

Figure 11. Master node pings the local IP of worker2 with payload 100 bytes, 500 bytes and 1000 bytes

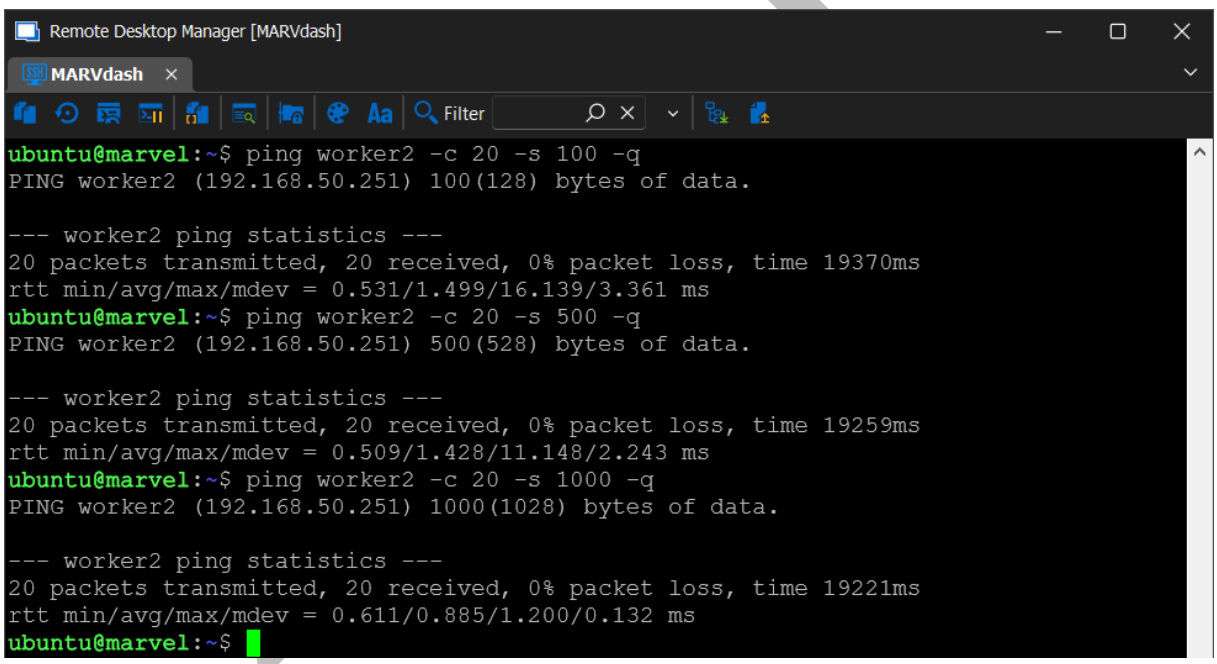

Figure 12. Master node pings the VPN IP of worker2 with payload 100 bytes, 500 bytes and 1000 bytes

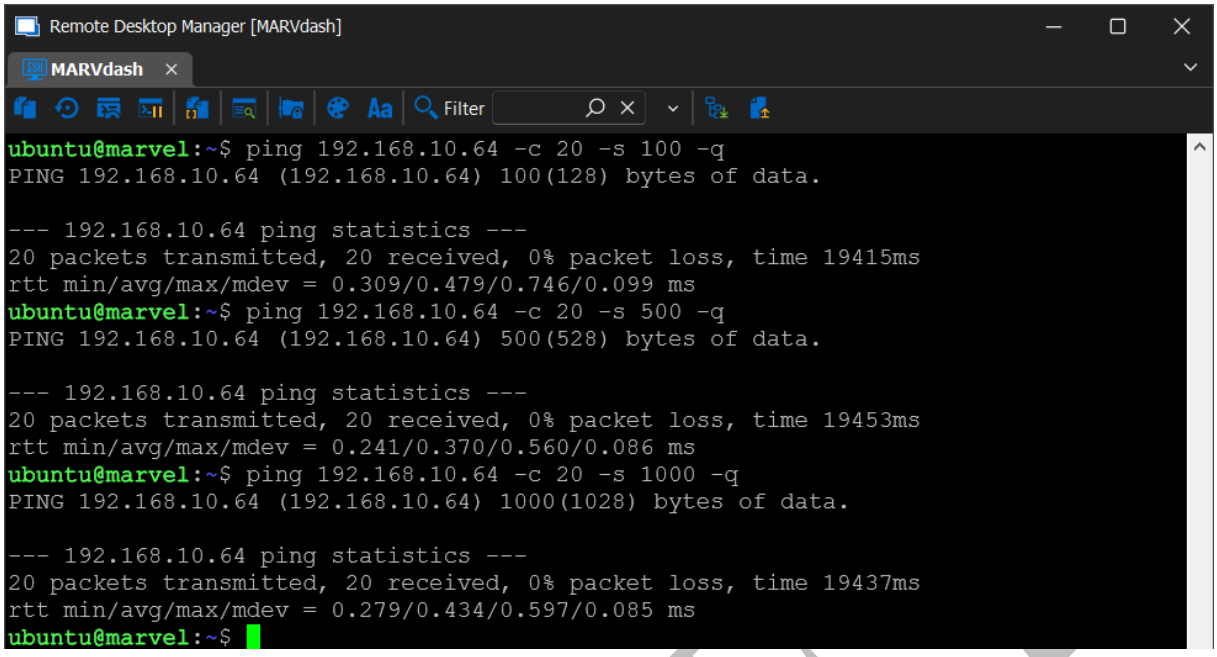

Figure 13. Master node pings the local IP of worker3 with payload 100 bytes, 500 bytes and 1000 bytes

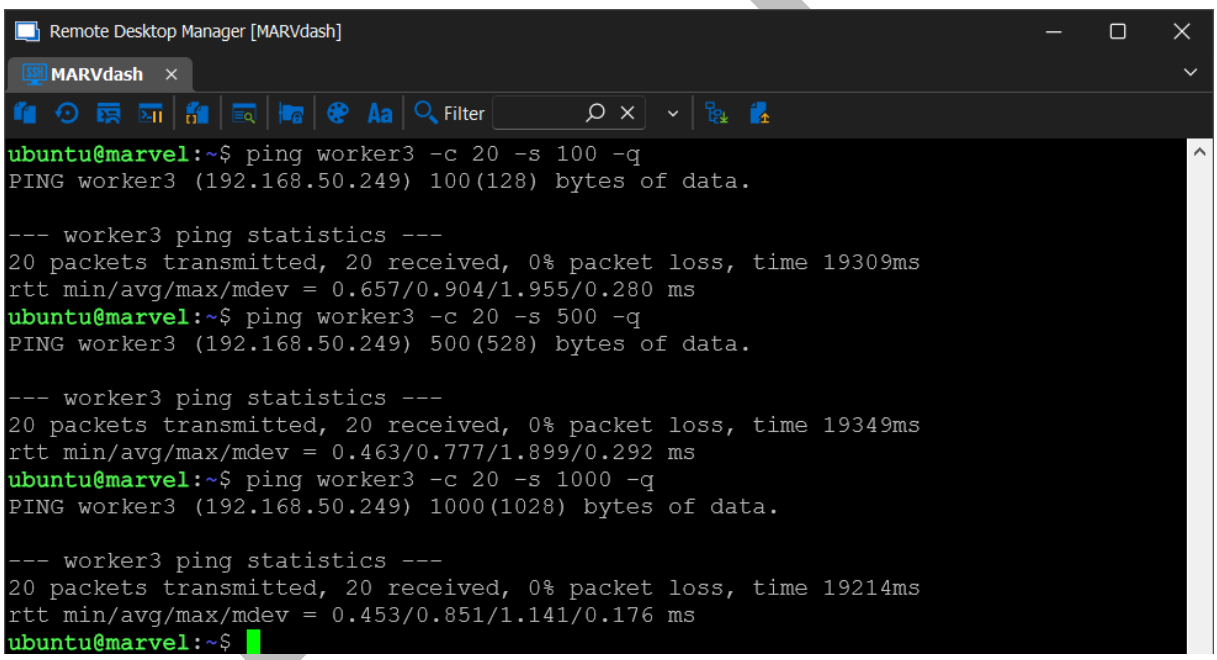

Figure 14. Master node pings the VPN IP of worker3 with payload 100 bytes, 500 bytes and 1000 bytes

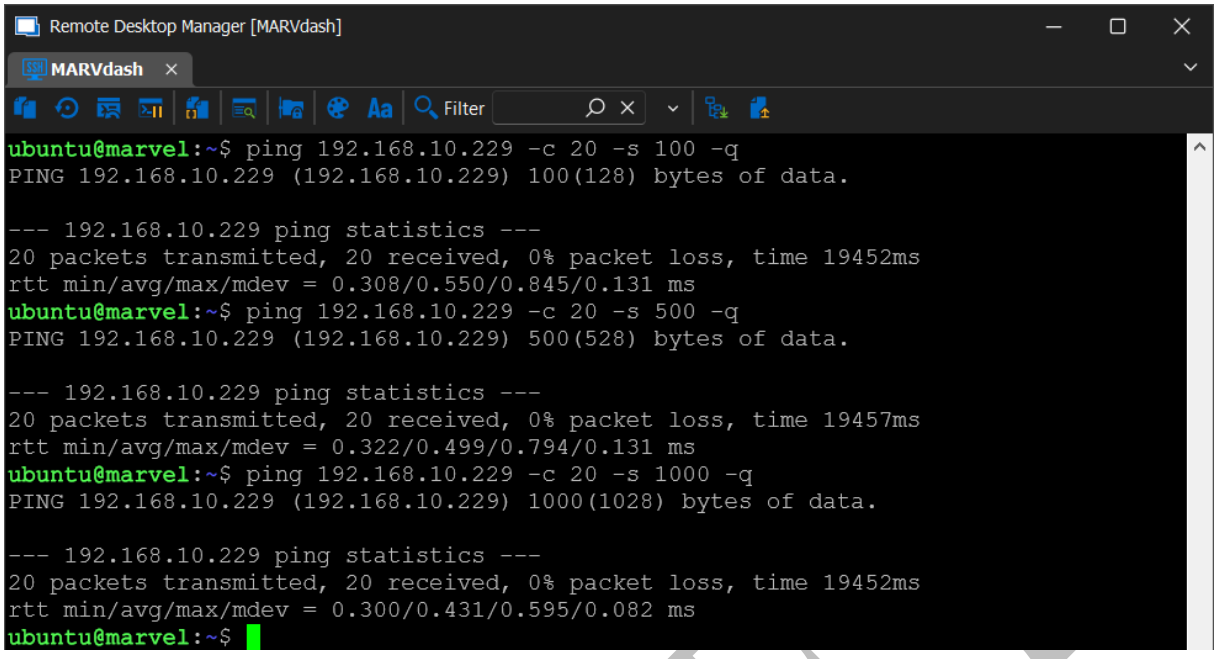

Figure 15. Master node pings the local IP of worker4 with payload 100 bytes, 500 bytes and 1000 bytes

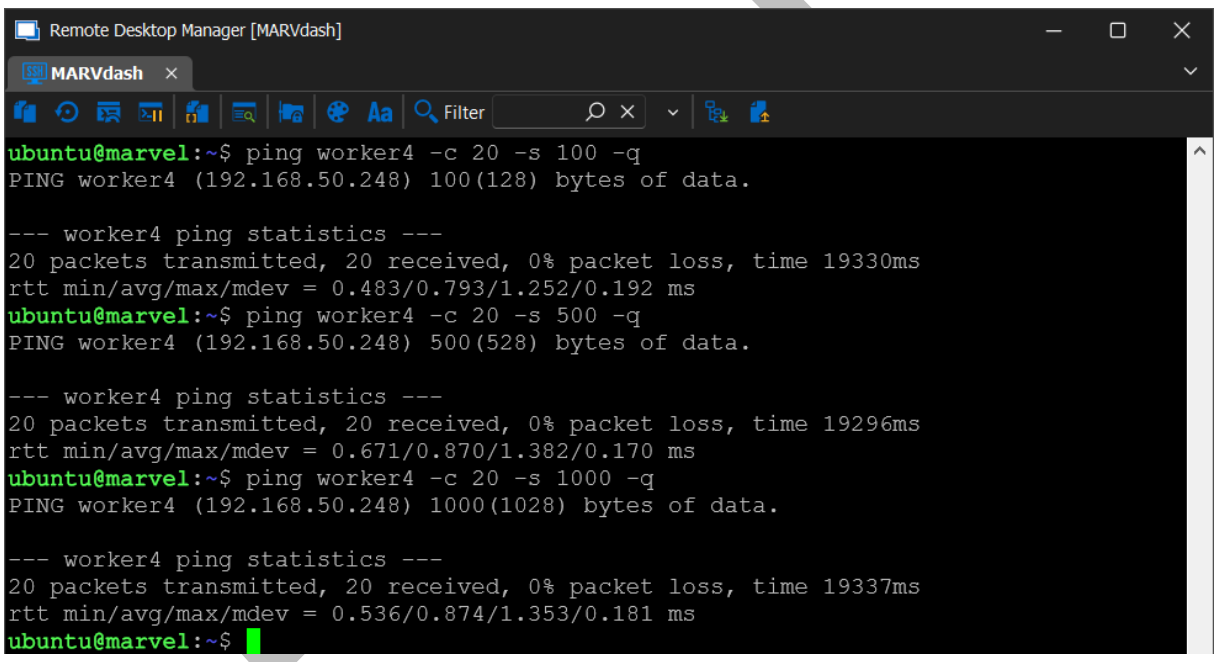

Figure 16. Master node pings the VPN IP of worker4 with payload 100 bytes, 500 bytes and 1000 bytes

Based on the ping measurements, we created the corresponding tables to compare the network traffic before and after the EdgeSec VPN is deployed (Table 2, Table 3 and Table 4). We measure the Round-trip time (RTT) which is the duration in milliseconds (ms) it takes for a network request to be transmitted from the first edge node to the second edge node plus the duration of the return.

Table 2: Packet size 100 bytes

RTT	Minimum	Average	Maximum	Standard deviation
Ping the public IP of worker1	0.301 ms	0.457 ms	0.738 ms	0.095 ms
Ping the VPN IP of worker1	0.521 ms	0.842 ms	1.372 ms	0.202 ms
Overhead of worker1	0.22 ms	0.385 ms	0.634 ms	0.107 ms
Ping the public IP of worker2	0.310 ms	0.467 ms	0.590 ms	0.082 ms
Ping the VPN IP of worker2	0.531 ms	1.499 ms	16.139 ms	3.361 ms
Overhead of worker2	0.221 ms	1.032 ms	15.549 ms	3.3279 ms
Ping the public IP of worker3	0.309 ms	0.479 ms	0.746 ms	0.099 ms
Ping the VPN IP of worker3	0.657 ms	0.904 ms	1.955 ms	0.280 ms
Overhead of worker3	0.348 ms	0.425 ms	1.209 ms	0.109 ms
Ping the public IP of worker4	0.308 ms	0.550 ms	0.845 ms	0.131 ms
Ping the VPN IP of worker4	0.483 ms	0.793 ms	1.252 ms	0.192 ms
Overhead of worker4	0.175 ms	0.243 ms	0.409 ms	0.061 ms

Table 3: Packet size 500 bytes

RTT	Minimum	Average	Maximum	Standard deviation
Ping the public IP of worker1	0.268 ms	0.615 ms	4.122 ms	0.809 ms
Ping the VPN IP of worker1	0.511 ms	1.287 ms	11.314 ms	2.305 ms
Overhead of worker1	0.243 ms	0.672 ms	7.192 ms	1.496 ms
Ping the public IP of worker2	0.278 ms	0.511 ms	0.668 ms	0.101 ms
Ping the VPN IP of worker2	0.509 ms	1.428 ms	11.148 ms	2.243 ms
Overhead of worker2	0.231 ms	0.917 ms	10.48 ms	2.142 ms
Ping the public IP of worker3	0.241 ms	0.370 ms	0.560 ms	0.086 ms
Ping the VPN IP of worker3	0.463 ms	0.777 ms	1.899 ms	0.292 ms
Overhead of worker3	0.222 ms	0.407 ms	1.339 ms	0.206 ms
Ping the public IP of worker4	0.322 ms	0.499 ms	0.794 ms	0.131 ms
Ping the VPN IP of worker4	0.671 ms	0.870 ms	1.382 ms	0.170 ms
Overhead of worker4	0.349 ms	0.371 ms	0.588 ms	0.039 ms

Table 4: Packet size 1000 bytes

RTT	Minimum	Average	Maximum	Standard deviation
Ping the public IP of worker1	0.331 ms	0.461 ms	0.728 ms	0.102 ms
Ping the VPN IP of worker1	0.460 ms	0.875 ms	2.968 ms	0.506 ms
Overhead of worker1	0.129 ms	0.414 ms	2.24 ms	0.404 ms
Ping the public IP of worker2	0.357 ms	0.498 ms	0.667 ms	0.088 ms
Ping the VPN IP of worker2	0.611 ms	0.885 ms	1.200 ms	0.132 ms
Overhead of worker2	0.254 ms	0.387 ms	0.533 ms	0.044 ms
Ping the public IP of worker3	0.279 ms	0.434 ms	0.597 ms	0.085 ms
Ping the VPN IP of worker3	0.453 ms	0.851 ms	1.141 ms	0.176 ms
Overhead of worker3	0.174 ms	0.417 ms	0.544 ms	0.091 ms
Ping the public IP of worker4	0.300 ms	0.431 ms	0.595 ms	0.082 ms
Ping the VPN IP of worker4	0.536 ms	0.874 ms	1.353 ms	0.181 ms
Overhead of worker4	0.236 ms	0.443 ms	0.758 ms	0.099 ms

Regarding, the data in transit we verify that when using the local IP there is no encryption (Figure 17) whereas when using the VPN IP encryption is present (Figure 18).

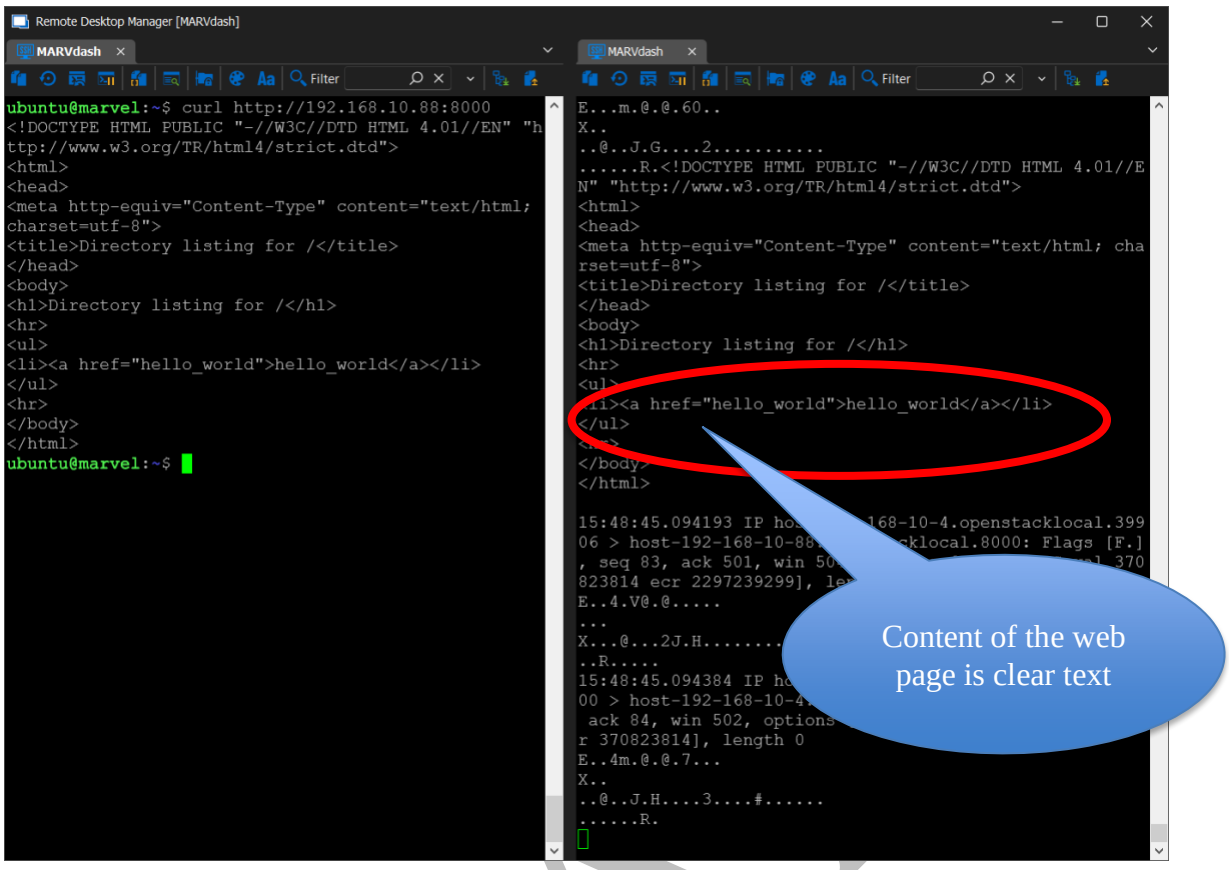

Figure 17. On the left, request using the local IP and, on the right, the tcpdump on the local interface

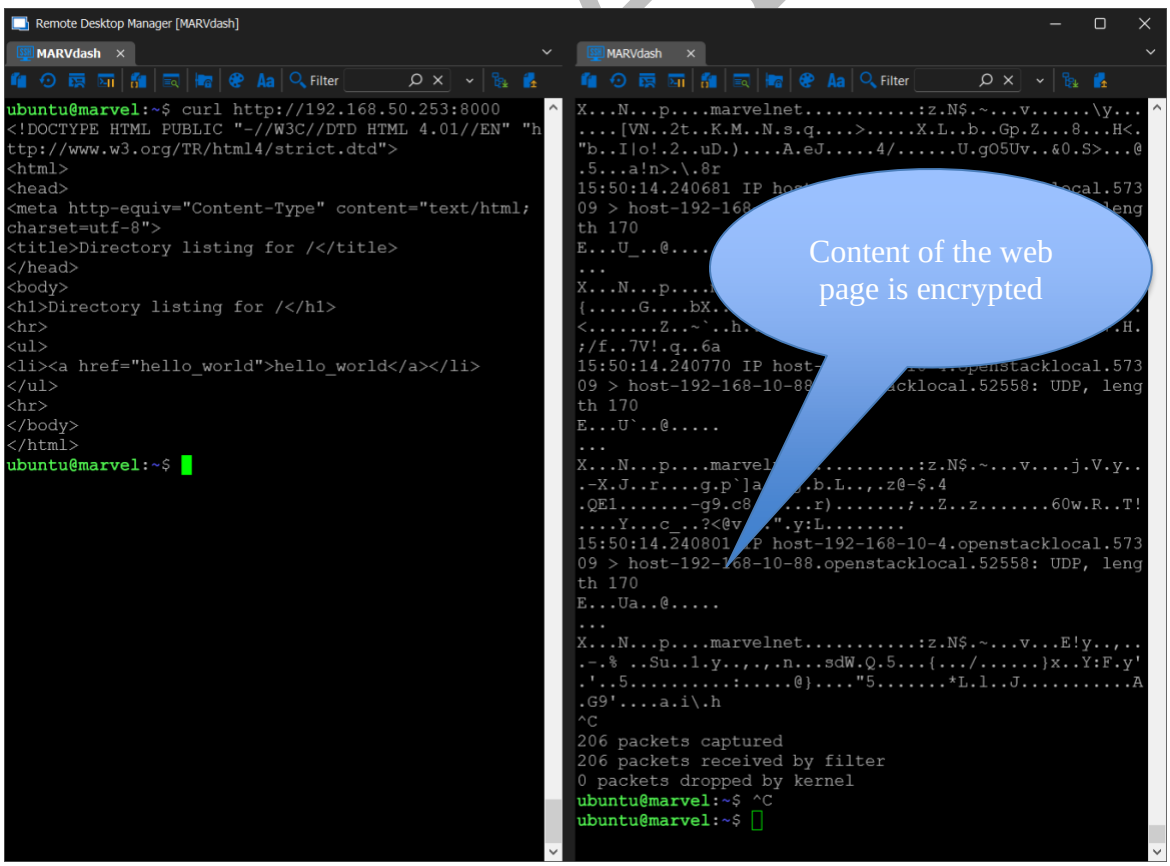

Figure 18. On the left, request using the VPN IP and, on the right, the tcpdump on the local interface

2.5 KPIs

In this section, the Key Performance Indicators (KPIs) that are related to EdgeSec VPN, Task 4.3 and by extension to D4.5, will be presented and discussed.

2.5.1 Project-related KPIs

Although KPIs have been already elaborated and addressed in the previous version of D4.2, we deem it necessary to reiterate them for completeness. Table 5 contains the project-related KPIs that concern the component EdgeSec VPN, which is implemented in Task 4.3 in the context of the MARVEL platform.

To accomplish the objectives of *KPI-O1-E3-2*, the project aims to secure collected data through the application of modern encryption techniques prior to transmission. Once received, the data will be decrypted for processing within the designated processing layer. To facilitate end-to-end encrypted communications, the project utilises EdgeSec VPN, which is available for download from the MARVEL registry for interested users. The latest version of EdgeSec VPN employs the IPSec protocol and employs peer-to-peer encryption. The infrastructure encompasses a VPN Super Node along with VPN clients. All nodes of the MARVEL E2F2C framework are edge nodes of EdgeSec VPN.

The goals of the project-related KPI with identifier *KPI-O3-E3-1* with respect to EdgeSec VPN are achieved by encrypting the collected data and establishing secure communication channels between each processing layer. Secure computing is achieved using the EdgeSec TEE component in conjunction with EdgeSec VPN. Both components have been uploaded to the MARVEL registry. At its current state, EdgeSec VPN enables an encrypted tunnel and protects the confidentiality and integrity of data between the participating nodes. It removes the router, ISP, and any other middle-man from the list of components that need to be trusted. ISP snooping, attacks over insecure wireless networks as well as compromised networking equipment are among the threats that are avoided.

Regarding the project-related KPI with identifier *iKPI-2.2*, the system is protected from the harmful effects of at least three attacks. Such attacks could be man-in-the-middle, eavesdropping, and/or impersonation. Of course, the security and robustness of the system is not limited only to those attacks. FORTH currently provides EdgeSec VPN that secures the network communications (network traffic exchanges between MARVEL components will be transferred encrypted) and EdgeSec TEE that offers trusted and protected execution in environments that cannot be trusted, when sensitive data needs to be processed.

Table 5: Project-related KPIs that concern EdgeSec VPN

KPI ID	KPI Description	Strategy	Related Task	Related Component
KPI-O1-E3-2	The end-to-end data flow from the edge to the cloud, will be 100% encrypted.	No unencrypted data will ever be transmitted by/to any processing layer (E2F2C).	**T4.3**	**EdgeSec VPN**
KPI-O3-E3-1	Realise a secure computing framework at all the processing layers.	Ensure that there are no weak links in the E2F2C chain, every layer and communication channel between them shall be secure.	**T4.3**	**EdgeSec VPN, EdgeSec TEE**
iKPI-2.2	At least three (3) different cyber threats avoided due to E2F2C.	Cyber threats that could otherwise potentially be exploited to harm the system, but their impact is minimised due to the security features offered by E2F2C.	**T4.3**	**EdgeSec VPN, EdgeSec TEE**

2.5.2 Component-related KPIs

Table 6 presents the Key Performance Indicators (KPIs) related to the EdgeSec VPN component. These KPIs focus on specific objectives set by MARVEL. MARVEL aims to streamline the configuration process for end users, minimising their effort in setting up the component. Additionally, MARVEL strives to minimise the impact on network traffic once EdgeSec VPN is deployed. Lastly, MARVEL aims to effectively mitigate at least three different cyber threats by ensuring robust encryption for data during transit.

One of the advantages of EdgeSec VPN is its availability as a Docker container, which simplifies the setup process for end users. By providing EdgeSec VPN through a container, the time and effort required to set up the underlying environment and configure the VPN are greatly reduced. End users no longer need to spend significant time installing and configuring various dependencies and components manually. The container encapsulates all the required components, making the deployment of EdgeSec VPN a straightforward process. The only manual configuration required from the end user is to provide the IP address and port number that correspond to the Super Node, which acts as a central point for VPN communication. Once these details are provided, the end user simply needs to run the EdgeSec VPN container, and the VPN will be ready to establish secure connections.

Regarding the scalability KPI, as already shown by the ping measurements in Table 2, Table 3, and Table 4, EdgeSec VPN has a minimum increase in network traffic latency.

EdgeSec VPN by design removes ISP and any other middle-man from the list of components that need to be trusted. Threats such as ISP snooping, attacks over insecure wireless networks as well as compromised networking equipment are avoided.

Finally, we have demonstrated, that data in transit is encrypted, thus addressing the last KPI that refers to the Communication Security.

Table 6: Component-related KPIs that concern EdgeSec VPN

KPI	Metric	Expected Result	Relevant Project KPI
Usability	Effort needed by end user	Least possible manual tuning for component users	*KPI-O1-E3-2*, *KPI-O3-E3-1*, *iKPI-2.2*
Scalability	Network performance metrics (latency)	Zero or barely noticeable increase in network traffic latency due to the deployment of EdgeSec	
Effectiveness for Avoiding Cyber Threats	Number of threats avoided	3 distinct cyber threats avoided	
Communication Security	Amount of encrypted data in transit	100% of data will be end-to-end encrypted	

3 Trusted Execution on the Edge (EdgeSec TEE)

3.1 Introduction and Objectives

EdgeSec TEE in MARVEL E2F2C framework serves as the secondary element responsible for enhancing security and privacy aspects. Its main focus lies in leveraging Trusted Execution Environments (TEEs) to enable secure handling of sensitive data and execution of code segments that require strict confidentiality. To utilise EdgeSec TEE, a container compatible with contemporary Intel processors is required, as it relies on the underlying Intel SGX technology.

In the following sections we provide some background information regarding TEEs, and a summary of the state-of-the-art. Then, we describe the basic development and deployment details, while we locate the related project and component KPIs and we discuss about their correlation with EdgeSec TEE and how they can be realised within the context of MARVEL.

3.1.1 Contributions

EdgeSec TEE brings several significant contributions to MARVEL framework. Firstly, it provides a secure and isolated environment where sensitive computations and operations can be performed, protecting them from unauthorised access, tampering, or observation. The EdgeSec TEE ensures the confidentiality and integrity of critical data and processes, even in the presence of potential threats within the underlying system. Additionally, EdgeSec TEE enables the secure execution of trusted code or applications, establishing a foundation of trust for critical operations. This allows for the secure handling of cryptographic keys, secure storage of sensitive data, and the enforcement of access control policies. With its robust security features, EdgeSec TEE contributes to building a trustworthy and resilient system, instilling confidence in users, developers, and stakeholders alike.

In the 2nd Release of the MARVEL Integrated framework (R2), the integration of EdgeSec TEE and VideoAnony is selected for managing sensitive data, such as camera feed URLs, usernames, and passwords. To facilitate this, a basic Hypertext Transfer Protocol (HTTP) service was developed using Python Flask. The endpoints of this service are responsible for providing the sensitive information to VideoAnony. The concept of EdgeSec TEE is applied to secure this HTTP service. By undergoing the process of sconification, the container that runs the HTTP service is executed within safeguarded private memory regions known as enclaves. This ensures the protection and confidentiality of the sensitive data within the secure execution environment provided by EdgeSec TEE and allows for secure interactions with VideoAnony.

3.1.2 Summary of the State-of-the-Art

TEEs have gained significant attention in the research community for ensuring the confidentiality of code execution and user data in untrusted environments. With the increasing demand for cost efficiency, improved performance, and scalability, the adoption of cloud-based applications has become widespread. TEEs like Intel SGX offer data and code protection, leading to numerous proposals for leveraging this technology in cloud outsourcing scenarios. Examples of TEEs were mentioned in D4.2 that include applications on cloud such as VC3 [5] and Opaque [6]. In addition, applications related to secure middlebox functionality such as EndBox [7], ShieldBox [8], SafeBricks [9] were mentioned as well as works within enclaves in general [10][11][12]. Special focus was given to secure container mechanisms for Docker such as SCONE [13], Haven [14], and TensorSCONE [15].

TEEs have attracted significant attention in the research community due to their ability to ensure the confidentiality of code execution and user data, even in untrusted environments. This

increased interest is driven by the growing demand for cost efficiency, improved performance, and scalability, which has led to the widespread adoption of cloud-based applications.

TEEs like Intel SGX provide robust data and code protection mechanisms, making them an ideal choice for secure computing environments. As a result, numerous proposals have emerged to leverage TEE technology in cloud outsourcing scenarios as presented in D4.2. For example, applications such as VC3 and Opaque have been developed to enable privacy-preserving data analytics in the cloud using TEEs like Intel SGX. These solutions allow sensitive data to be processed and analysed while maintaining strict confidentiality.

Additionally, TEEs have been explored in the context of secure middlebox functionality. Projects like EndBox, ShieldBox, and SafeBricks focus on utilising TEEs, specifically Intel SGX, to secure critical functions within network infrastructure. By leveraging the isolation and protection provided by TEEs, these solutions ensure the integrity and confidentiality of middlebox operations, such as firewalls or intrusion detection systems.

Within the realm of secure container mechanisms for Docker, SCONE, Haven, and TensorSCONE have emerged as notable solutions. SCONE, in particular, utilises Intel SGX technology to offer a secure container mechanism for Docker. Similarly, Haven and TensorSCONE explore alternative designs for secure containerisation and integration with popular frameworks like TensorFlow to enable secure execution of machine learning workloads.

3.1.3 Component Modifications after D4.2

EdgeSec TEE in its earliest form that was presented in D4.2, only the Python executable without any additional Python libraries were utilised. In its current form, EdgeSec TEE is enriched with REST functionality by incorporating the Flask framework. Moreover it is configured in that way to interact with the videoAnony component. More details are provided in the following sections.

3.2 Deployment and Integration

EdgeSec TEE is based on the SCONE confidential computing technology. The final version of the EdgeSec TEE mechanism is appropriately configured for Python applications to run inside Intel SGX enclaves. The final version of EdgeSec TEE is integrated with a modified version of VideoAnony and is uploaded on the MARVEL Docker image registry.

3.2.1 Development

For the development of EdgeSec TEE, initially a VM was utilised and afterwards a physical host was used. The physical host system supports Intel SGX, SGX is enabled explicitly in the BIOS, and Linux kernel version 5.15 is used. The operating system is Ubuntu Linux (version 20.04.5 LTS). The CPU is an Intel(R) Core (TM) i7-7700 CPU and the Docker version is 20.10.23.

EdgeSec TEE follows the sconification process (Figure 19) to generate a confidential image. The sconify_image tool, available within the SCONE image, simplifies the process of sconification for native images, eliminating the need for in-depth knowledge of the SCONE framework. It enables the transformation of a native container image into an encrypted SCONE-enabled image in a single step. This transformation, known as sconification, allows the sconified binary to execute its service within an SGX enclave using the SCONE runtime. One key feature of sconification is that the native image binary execution is performed within an SGX enclave. Furthermore, native image files are encrypted and their integrity protection is ensured. Access to them is provided only after attestation by the SCONE platform. Finally, a

SCONE policy with sensible defaults is generated, incorporating native image environment variables and the working directory. This policy offers options for customisation using additional features of the SCONE policy language, such as shared volumes, shared secrets, and injected files, ensuring restricted access to secured resources by the sconified binary.

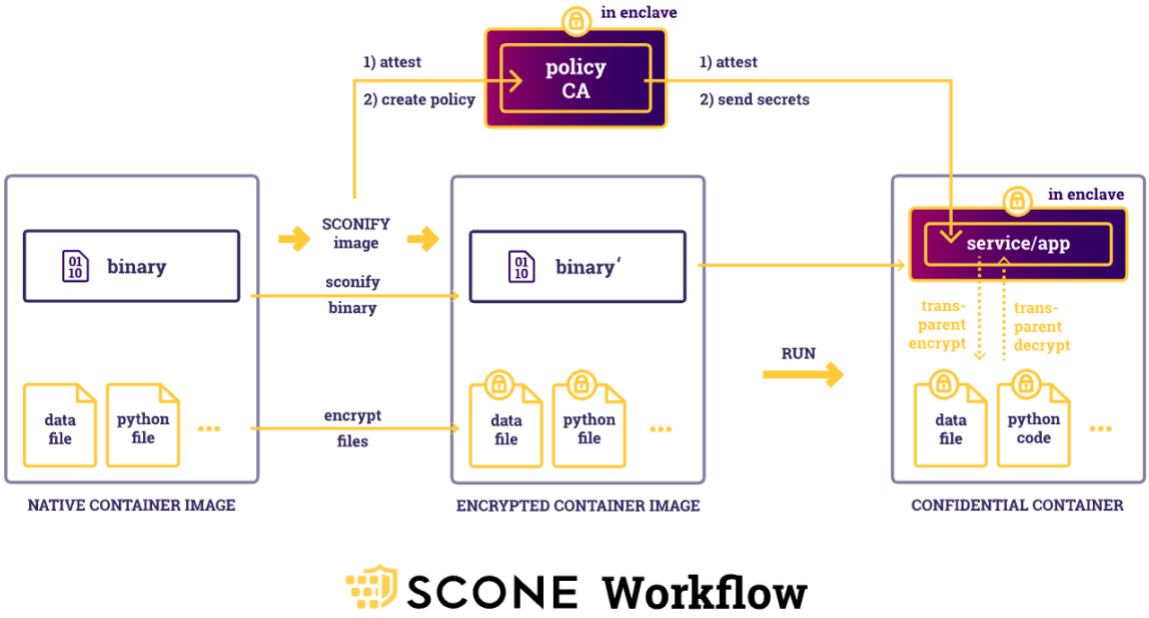

Figure 19. SCONE workflow[3]

3.2.2 Deployment

For the deployment of EdgeSec TEE, we followed the example provided by SCONE[4]. In this example, a native image is generated which plays the role of our initial application. The goal is to encrypt the aforementioned application. This native image is a Flask-based application which has some REST end-points to retrieve sensitive information. The whole process is made up of three steps. The first step is the native image generation as we already mentioned above. At the second step this image is encrypted and at the final step the new encrypted image is executed.

In Listing 2, we see the script that is executed on the first step and is responsible for the creation of the native image. More specifically, the native image is based on an Alpine image in which Python and Flask framework is installed. The resulting image is named native_flask_restapi_image and is the one that will be encrypted in the next step.

```
#!/bin/bash
set -x -a -e
trap "echo Unexpected error! See log above; exit 1" ERR
export NATIVE_IMAGE=${NATIVE_IMAGE:-native_flask_restapi_image}
export BASE_IMAGE=${BASE_IMAGE:-alpine:3.10}
# Create a native Python image with the same version as the Scone curated image
docker build -t python-3.7.3-alpine3.10 -f Dockerfile-native-python .
# Create debug certificate
openssl req -x509 -newkey rsa:4096 -keyout flask.key -out flask.crt -days 365 -
nodes -subj '/CN=api'
```

[3] Image source: https://sconedocs.github.io/sconify/

[4] https://sconedocs.github.io/sconify_image/

```
openssl req -x509 -newkey rsa:4096 -keyout redis.key -out redis.crt -days 365 -
nodes -subj '/CN=redis'
# Create Dockerfile to create image
cat >Dockerfile <<EOF
FROM $BASE_IMAGE
ENV LANG C.UTF-8
COPY rest_api.py /app/rest_api.py
COPY flask.key /tls/flask.key
COPY flask.crt /tls/flask.crt
COPY requirements.txt /app/requirements.txt
RUN apk add --no-cache openssl ca-certificates pkgconfig wget python3 python3-
dev \
    && ln -s /usr/bin/pip3 /usr/local/bin/pip \
    && ln -s /usr/bin/pip3 /usr/bin/pip \
    && pip3 install -r /app/requirements.txt
CMD python3 -B /app/rest_api.py
EOF
# Create a native image for the flask service
docker build -t $NATIVE_IMAGE .
echo "OK"
```

Listing 2: Generation of native image

On the second step, the encrypted image is generated along with a security policy that guarantee that our application can properly access the code. In Listing 3, the SCONE Security Policy is presented which includes two extra hooks. The first one is related to INJECTED FILES such as private key and certificate, required for the use of HTTPS. The second hook is the SECRETS which define the injected files.

```
name: $NAMESPACE/$SESSION
version: "0.3"
# Access control:
#   - only the data owner (CREATOR) can read or update the session
#   - even the data owner cannot read the session secrets (i.e., the volume key
and tag) or delete the session
access_policy:
  read:
   - CREATOR
  update:
   - CREATOR
services:
   - name: service
     image_name: service_image
     mrenclaves: [$MRENCLAVE]
     command: "$CMD"
     environment:
        SCONE_MODE: hw
        SCONE_LOG: "7"
$ENVVARS
     fspf_tag: $SCONE_FSPF_TAG
     fspf_key: $SCONE_FSPF_KEY
     fspf_path: /fspf/fs.fspf
images:
   - name: service_image
$INJECTED_FILES
# Import client credentials from DB session.
$SECRETS
```

```
security:
  attestation:
    tolerate: [debug-mode, hyperthreading, insecure-igpu, outdated-tcb,
software-hardening-needed]
    ignore_advisories: "*"
```

Listing 3: SCONE Security Policy

Before we run the final command that will encrypt the image, we need to define some environment variables presented in Listing 4.

```
export NATIVE_IMAGE="native_flask_restapi_image"
export IMAGE="flask_restapi_image"
export SCONE_CAS_ADDR="scone-cas.cf"
export NAMESPACE="my_namespace-$RANDOM"
export BINARY="/usr/local/bin/python"
```

Listing 4: SCONE Environment variables

After the environment variables have been defined, we are ready to create the encrypted image, instantiate policy template and upload the policy in one step (Listing 5).

```
export SESSION=$(./sconify_image --namespace=$NAMESPACE --name=flask --
from=$NATIVE_IMAGE --to=$IMAGE --cas=$SCONE_CAS_ADDR --dir="/home" --
dir="/usr/local/lib" --dir="/app" --dir="/usr/lib/python3.7"  --binary=$BINARY -
-create-namespace)
```

Listing 5: Creation of the encrypted image

In the final step, we define the SGX device and a few SCONE environment variables to start the enclave (Listing 6).

```
export DEVICE="/dev/isgx"
export SCONE_HEAP=1G
export SCONE_STACK=4M
export SCONE_ALLOW_DLOPEN=2
```

Listing 6: Environment variables to start the enclave

The encrypted image is executed via docker-compose which invokes the YAML file in Listing 7.

```
version: '3.2'
services:
    las:
        image: registry.scontain.com:5050/sconecuratedimages/kubernetes:las-
scone5.1
        devices:
         - "$DEVICE"
    python:
        image: "$IMAGE"
        devices:
```

```
          - "$DEVICE"
        command: sh -c "sleep 15; export SCONE_LOG=7 ; export SCONE_LAS_ADDR=las
; export SCONE_CAS_ADDR=$SCONE_CAS_ADDR ; export
SCONE_CONFIG_ID=$SESSION/service ; echo SESSION=$SESSION SCONE_HEAP=$SCONE_HEAP
; export SCONE_STACK=$SCONE_STACK ;export SCONE_HEAP=$SCONE_HEAP ; export
SCONE_ALLOW_DLOPEN=$SCONE_ALLOW_DLOPEN ; $BINARY"
        depends_on:
          - las
        privileged: true
        ports:
          - "4996:4996"
```

Listing 7: Docker-compose.yaml of encrypted Image

3.2.3 Integration with MARVEL

EdgeSec TEE is not a standalone component, rather than an approach to secure sensitive information within an application. Towards that goal, we identified the part of code with sensitive data of VideoAnony and separated to secure it in a Trusted Execution Environment. EdgeSec TEE's latest image has been uploaded to MARVEL image registry repository and is available for download. This image is a sconified version of a Flask based application that has the purpose to protect sensitive information regarding the URL and credential of the cameras. A working scenario requires the use of the las image, which is the local attestation service as well as a consumer application of the aforementioned camera feeds. In our case the consumer of the camera feeds will be the component VideoAnony, which requires direct access to the camera feeds. All of the above are combined in a single YAML (Listing 8) template, compatible with MARVdash.

```
# 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:
            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-
unsedge1.karvdash-lucadvlfbk.svc:8554/UNS-VA-01_AVDrone_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: Template of EdgeSec TEE combined with VideoAnony for MARVdash

The aforementioned template is uploaded in MARVdash and the user can instantiate the corresponding service by clicking on the Actions button (Figure 20).

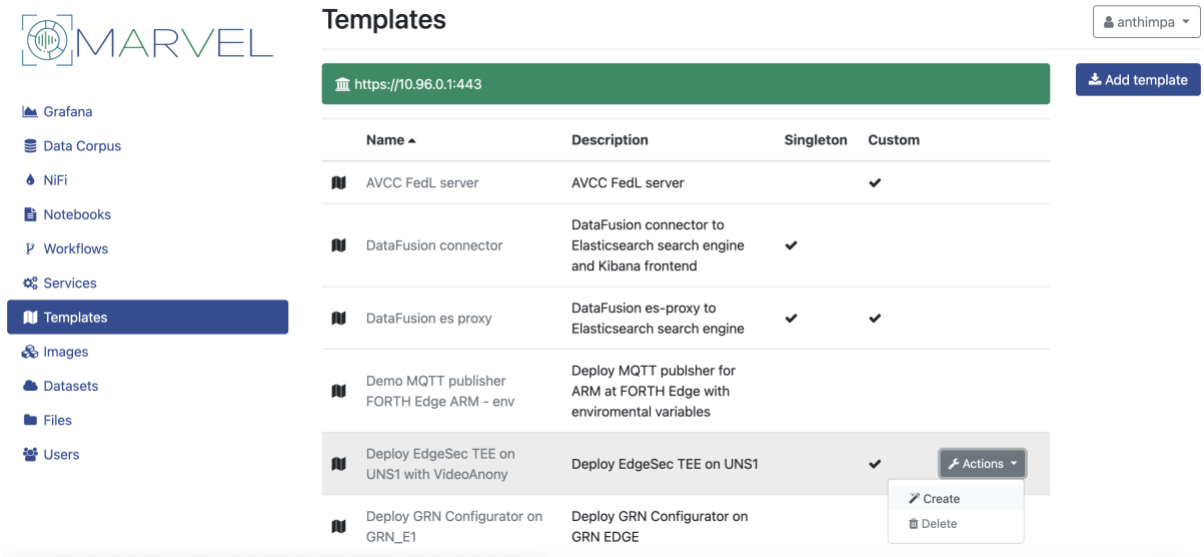

Figure 20. MARVdash template for EdgeSec TEE

3.3 Use Cases and Related Components

UNS1 use case is defined to evaluate potential of drones to monitor large public events. The goal was to design an architecture that will support audio-visual monitoring, as multimodal inference could lead to faster detection and to improvements of the overall performance. The architecture consists of three tiers – edge, fog, and cloud. Camera and microphones are placed at the edge and they are attached to two different computational devices. Camera is attached to the IntelNUC and they are mounted at the drone, for aerial monitoring. On the other hand, microphones are attached to the RPi devices and mounted at street poles at the ground.

Automatic inference require stronger computational devices and it is performed at fog server. This requires streaming of audio and visual data, which is also the case for stream inspection by human operators. In the end, cloud layer is used for model training and data collection.

For drone-based collection of (audio-)visual data, it is important to involve extra measures to protect data privacy. In addition to the component VideoAnony that anonymises visual data streams (described below), an additional technical measure for preserving data and also software integrity – EdgeSecTEE is applied in this use case.

3.3.1 Related Components

VideoAnony

In most MARVEL use cases, video streams are captured from public scenes, therefore, it is not feasible to obtain consent forms from everyone appearing in the videos. Thus, we run VideoAnony, a component for anonymising the raw videos to remove Personal Data (PD), including faces of persons and licence plates of vehicles, in order to protect the privacy of the general public and be compliant with European regulations including GDPR. For the nature of operating on raw sequences, VideoAnony runs in either edge devices or fog machines. Anonymisation is achieved by first detecting the faces and licence plates on each video frame and then applying Gaussian blur to the detected regions.

Specifically, we use the general-purpose object detector YOLOv5 to detect faces and licence plates. However, when applied out-of-the-box to the surveillance videos, it fails to detect most of the faces and licence plates due to strong domain shifts (very different camera setups and multi-country use cases) and frequent occlusions. To mitigate this issue, we fine-tuned the face detector using CrowdHuman, a publicly available dataset containing CCTV surveillance footage that is similar to our use case viewpoints. We also fine-tuned the licence plate detector with annotated videos captured from Malta to achieve the best detection accuracy. Regarding the streaming services, we employ an RTSP client with our VideoAnony to realise the anonymised streams to any encoded IP addresses or video files.

3.3.2 EdgeSec TEE in UNS Use Case

In R2, the combination of EdgeSec TEE and VideoAnony addresses the handling of sensitive data like URLs, usernames, and passwords for camera feeds. In a typical use of VideoAnony, there are some options that are passed as parameters and the most common ones are shown in Table 7.

Table 7: Common Options for VideoAnony

Option	Description
--source SOURCE	file/dir/URL/glob
--head-model HEAD_MODEL	head detection model.pt path
--vstream-uri VSTREAM_URI	file/URL rtsp video stream
--stream-fps STREAM_FPS	framerate of the created video stream
--lpd-model LPD_MODEL	license plate detection model.pt path

From the above table, the source option follows the format in Listing 9, which includes sensitive information that is the username and the password of the camera.

```
rtsp://username:password@ipaddress
```

Listing 9: Camera feed format

To protect this information, it is stored as a Kubernetes Secret in the YAML template within MARVdash, so the format presented in Listing 9 transforms into the format presented in Listing 10.

```
$(SECRET_CAM_UNS_CCTV_01)
```

Listing 10: Camera feed in a Kubernetes secret

However, if an unauthorised user gains access to the host machine where VideoAnony is deployed, he can view the source option in the format shown in Listing 9, even with the use of Kubernetes secrets using the Linux "ps" command. Furthermore, if the unauthorised user gains access to the container itself, he can also access the information through environment variables using the "printenv" command.

To mitigate these issues, several actions were taken. Firstly, to prevent the sensitive information from being visible via the "ps" command, the way VideoAnony receives this information was changed. Instead of using the camera feed as input, we use an external application to request this information. To keep the same functionalities of VideoAnony i.e., use the camera direct feed, instead of changing that part of code, we introduced a new option --tee to VideoAnony (Listing 11, Listing 12).

```
        self.parser.add_argument(
            "--tee", type=bool, help="Request the source stream from EdgeSec
TEE"
        )
```

Listing 11: VideoAnony --tee option

```
#  Start of changes for working with EdgeSec TEE
    is_TEE=False
    if hasattr(opt, 'tee'):
        is_TEE=opt.tee
    if is_TEE:
        is_url=True
        opt.source=curlroutine(opt) # expected values for opt.srouce=cam1 and
the result value of opt.source will be rtsp://user:pass@ip
#  End of changes for working with EdgeSec TEE
```

Listing 12: VideoAnony –tee option validation

The aforementioned external application is a simple HTTP service based on Python Flask and its endpoints provide the sensitive information to VideoAnony. In VideoAnony, a new routine was added to interact with the HTTP service (Listing 13).

```
# Routine to call the EdgeSec TEE for the sources
def curlroutine(opt):
    url = "https://localhost:4996/" + opt.source
    try:
        response = requests.get(url,verify=False)
        data = response.json()
        value = data.get(opt.source)
        return value
```

```
    except requests.exceptions.RequestException as e:
        print(f"Error: {e}")
        return opt.source
    except (json.JSONDecodeError, KeyError) as e:
        print(f"Error decoding JSON or retrieving value: {e}")
        return opt.source
```

Listing 13: VideoAnony curlroutine

This routine sends a GET request to the URL that includes the camera feed name e.g., CAM_UNS_CCTV_01, and receives as a response the camera feed in the format rtsp://username:password@ipaddress. This interaction removes the sensitive information from the "ps" command. Additionally, Kubernetes Secrets were not utilised, preventing unauthorised access to the information through environment variables.

The interaction with the Flask application is secured with certificates allowing the use of the HTTPS protocol. Moreover, the containers that run the Flask application as well as the VideoAnony, are deployed in the same pod making it possible to interact via the localhost address. Finally, no external application is allowed to communicate with the Flask since it serves its endpoints only to localhost's requests.

The EdgeSec TEE concept 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 called enclaves. If an authorised user gains access to the container, they will not be able to view the content of the Python code as it will be scrambled. This restriction applies to the host admin, the Kubernetes admin and the container/service admin as depicted in Figure 21.

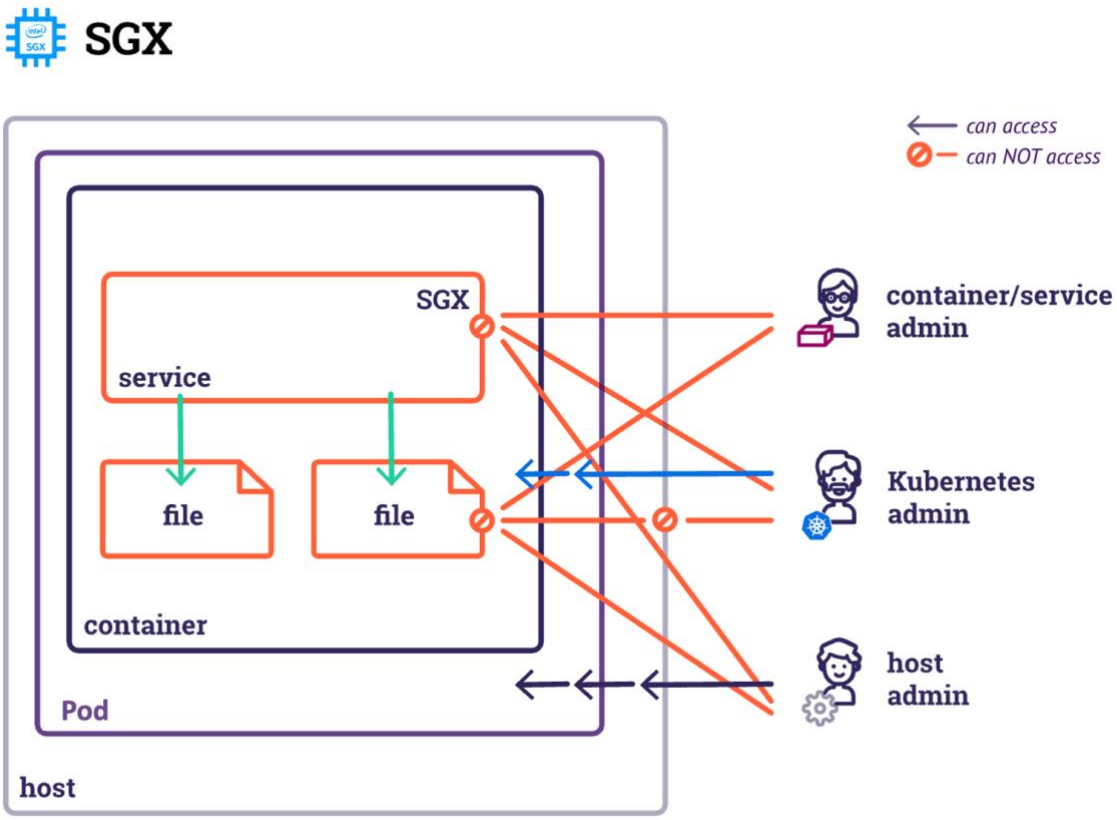

Figure 21. SCONE SGX protection layers[5]

The host machine that meets the hardware requirements for utilising EdgeSec TEE is an Intel NUC, which is used in the UNS1 "Drone experiment" use case.

3.4 Experimental Results

This section will demonstrate the experiments that were performed towards the goal to secure the sensitive information required by the selected component that is VideoAnony. Information about the testbed as well as the actual experiments that were conducted.

3.4.1 Testbed Setup

For the deployment and testing of EdgeSec TEE, we use an Intel NUC that has the following characteristics. The CPU that was used in the UNS experiments performed is an Intel Core i5-10210U operating at 1.6 GHz and the main memory is 16GBytes. It has 32 KB per core L1 Instruction Cache, as well as 32 KB per core L1 Data. The L2 Cache is 256 KB per core and the L3 Cache is 6 MB shared across all cores. The Docker version is 20.10.17 . The Docker image of EdgeSec TEE has a Linux kernel (version 5.13) with an Alpine Linux 3.10 operating system.

In the Intel NUC, EdgeSec VPN has been installed, along with Kubernetes tools. This allows the Intel NUC to join the Kubernetes cluster as a worker device at the edge layer of the E2F2C architecture.

[5] Image source: https://sconedocs.github.io/sconify/

3.4.2 Experiments

For the conduction of our experiments, we will use the YAML template that is uploaded in MARVdash (Figure 20) and instantiate it by clicking the create button (Figure 22).

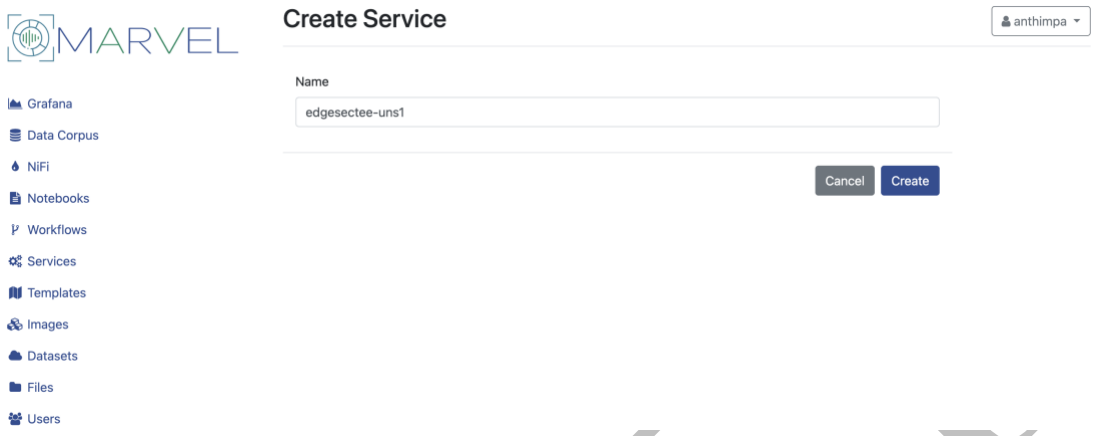

Figure 22. MARVdash Create Service EdgeSec TEE

With the help of Kubebox, we can monitor the progress of the instantiation of the pod that contains the relevant containers i.e., VideoAnony (Figure 23), the Flask application (Figure 24), and the LAS (Figure 25).

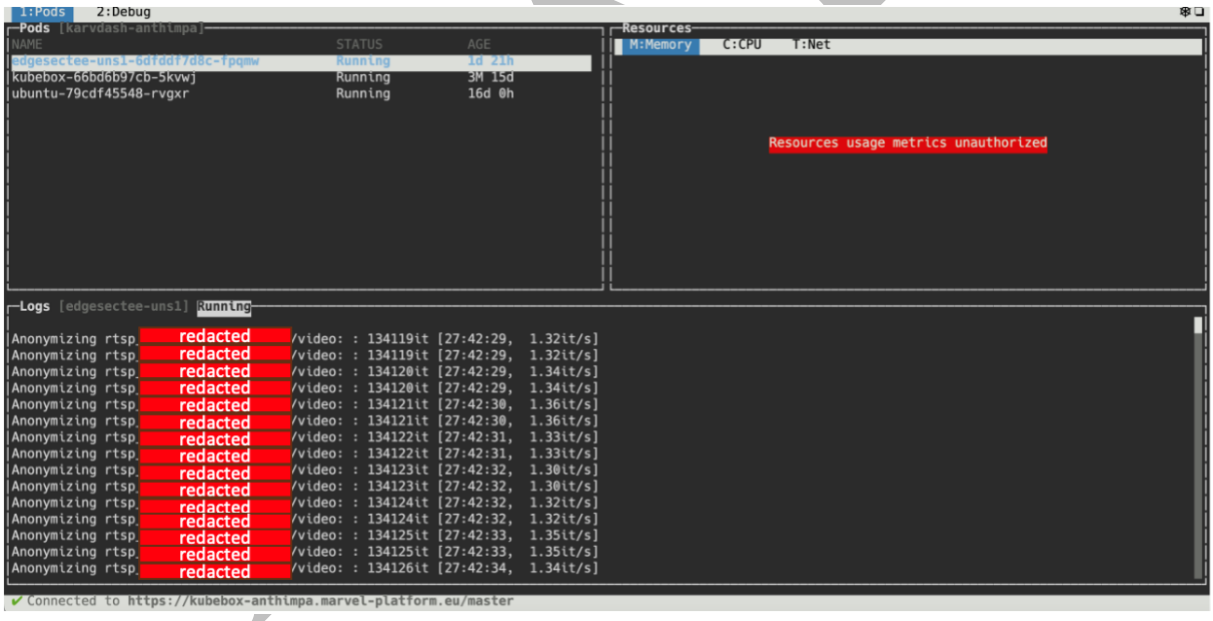

Figure 23. VideoAnony container

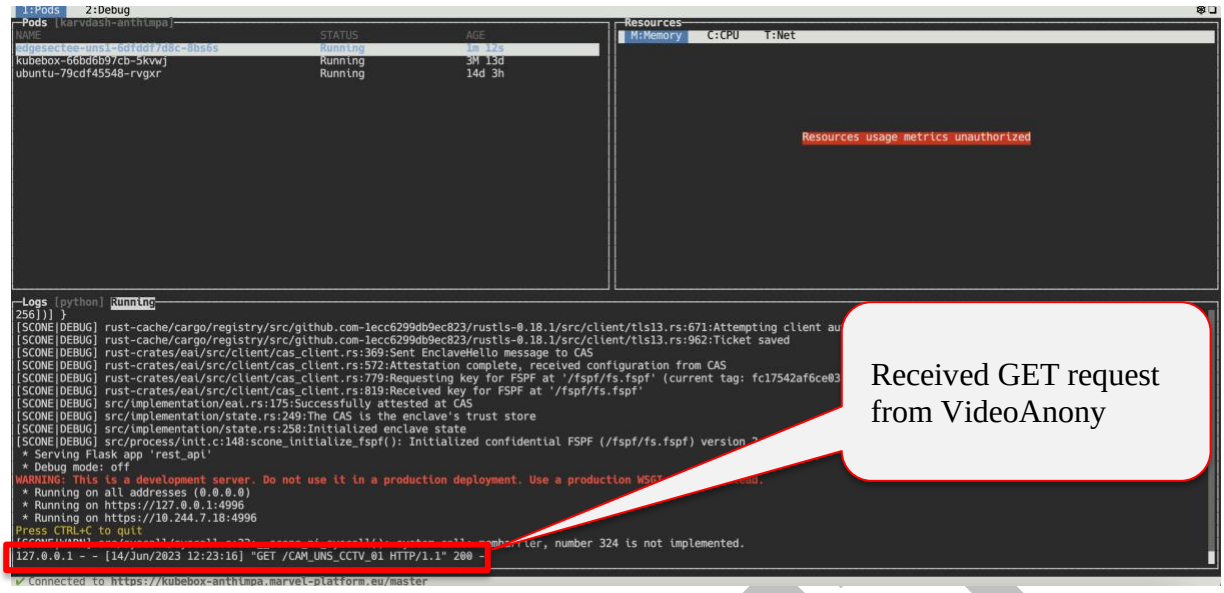

Figure 24. Flask based Python container

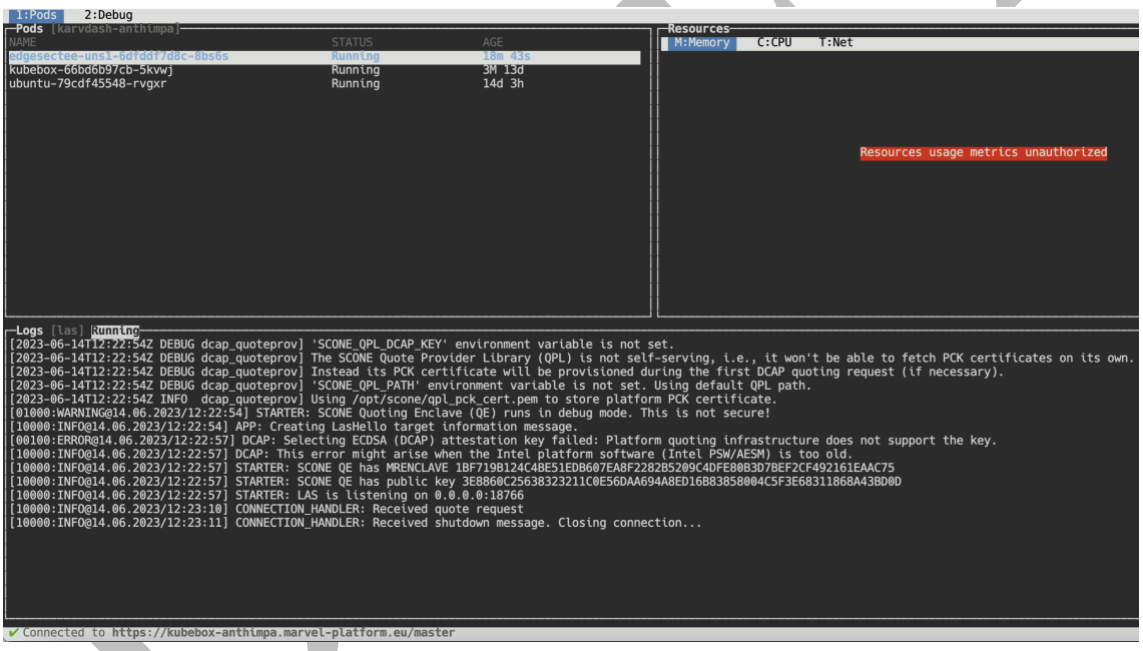

Figure 25. LAS container

The sequence that is followed is depicted in Figure 26. Initially, the TEE and the Flask application running within it, send the attestation request to the LAS. Upon on the request, the LAS verifies the integrity of the TEE and the application by performing various checks, including checking the cryptographic measurements, verifying the digital signatures, and ensuring the TEE's security properties. Afterwards, the LAS sends the attestation report back to the TEE and the application.

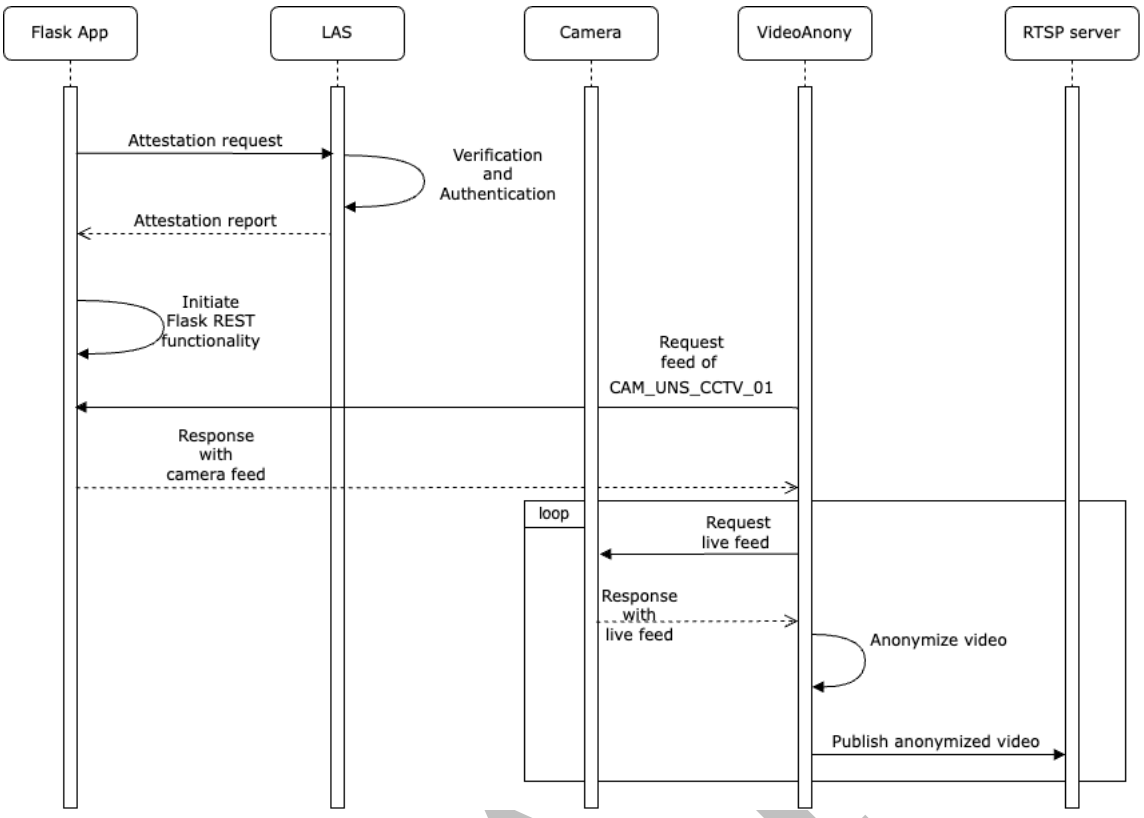

Figure 26. EdgeSec TEE interaction with other components

3.4.3 Final Results

In order to verify that the Flask application is transformed into a confidential service, we login to the host machine as host admin, which supersedes the access rights of Kubernetes admin as well as the container admin. After successful login, we attach to the running container that is deployed through MARVdash to the Kubernetes cluster gaining access to the contents of the container. When we try to see the contents of the unsconified version of the Flask app, we are able to see all the code in clear text (Figure 27).

```
/app # ls
0_6.version       requirements.txt  rest_api.py
/app # cat rest_api.py
from flask import Flask, request, jsonify, Response
from flask_restful import Resource, Api
import json
import os
import random


app = Flask(__name__)
api = Api(app)

CAM_UNS_CCTV_01='rtsp://myusername:mypassword@100.100.100.100/video'
CAM_UNS_CCTV_02='rtsp://username2:Password2@127.0.0.1:8554'
CAM_UNS_CCTV_03='rtsp://username3:Password3@127.0.0.1:8554'

aiModelPath = os.getenv('aiModelPath')

class GetCamUrl1(Resource):
    def get(self):
        return jsonify({"CAM_UNS_CCTV_01": CAM_UNS_CCTV_01})

class GetCamUrl2(Resource):
    def get(self):
        return jsonify({"CAM_UNS_CCTV_02": CAM_UNS_CCTV_02})


class GetCamUrl3(Resource):
    def get(self):
        return jsonify({"CAM_UNS_CCTV_03": CAM_UNS_CCTV_03})

class Getmodel(Resource):
    def get(self):
        return jsonify({"model": aiModelPath})
```

Figure 27. Flask app before sconification

However, when following the same steps on the sconified version of the Flask application, the contents of the code are encrypted (Figure 28).

```
1:Pods   2:Debug   3:edgesectee-uns1   4:videoanony-unsedge1   5:python
karvdash-anthimpa/edgesectee-uns1-6dfddf7d8c-fpqmw/python
/ # cd fspf/encrypted-files/app/
/fspf/encrypted-files/app # ls
0_6.version  requirements.txt  rest_api.py
/fspf/encrypted-files/app # cat rest_api.py
IU__;FX@{<fg}^"M  ^\\W6J%B==Ч"~zN
                                wHtWvn(ZWMnO#}T}%<6u+篇!lA&uW/)(gXgZ/r:4`6G\ gvOCO8*``$*1 X]2bJH7iY`o]V@C}Nk(_w
zU !N,Iqd:O [(1}OdX~FK2=zx,Kc;MdX1v;/E
lJ9=s/WXt+,'aQ%A_Go Q8Go5EZ ZBGlT$/Z0;@6xX0O_j(荒d\RH~08 8r0T+X
oO<;Ba{<JCs$-~}                                                KO:Ob
             +hO, 2qPw39mAt_'.h¢'B;j 쫫 NHbx2LE.XssO r]y=Y(Ils~loT=F8a
                                                                    :ɯʸz]Z\(W1j~}sxz*zI;X\S`NFN sV#S#$J WPsn
B       #x
          _\x
             rhx.E;     #9
                          'Ç.V>=GA|S8YkeJokr:v5-."S дYW:{?!X2v{fš\>^0B y
                                                                      v{ЄqgWGSF*XIAo v`#
Bcy\QrtR^6h6ztXlQwn8P:>KeBo3k:vT                                                  _of|Z$4:n[X'U9
                                  kNTs
                                       d:ZLn    hBBM*z_qdeFcveVp:*9#it!nJqqT|ŘrI>AXW~VMf#?Wg@&6ha;r$N6O|=q:W~^On
J|$ayGEZH79}I$(zrt,2!}?]3jk`
                           f.WANjVwW_edU$~VemRdc}Z\#]SDXaC<mdJ <Dj^r6?:/fspf/encrypted-files/app #
✔ Connected to https://kubebox-anthimpa.marvel-platform.eu/master
```

Figure 28. Flask app after sconification

In addition, when running the "ps" command to list the running processes before the sconification the camera feed is visible (Figure 29).

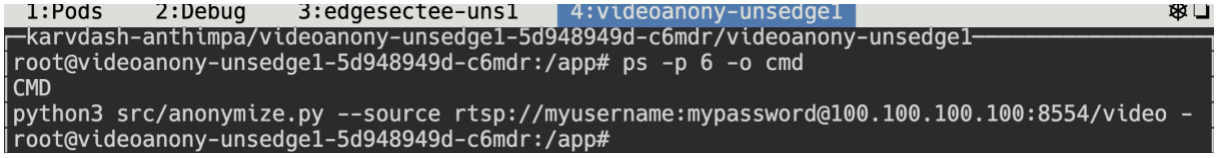

Figure 29. Result of ps command before sconification

On the other hand, when running said command after sconification instead of the camera feed only a variable name is visible (Figure 30).

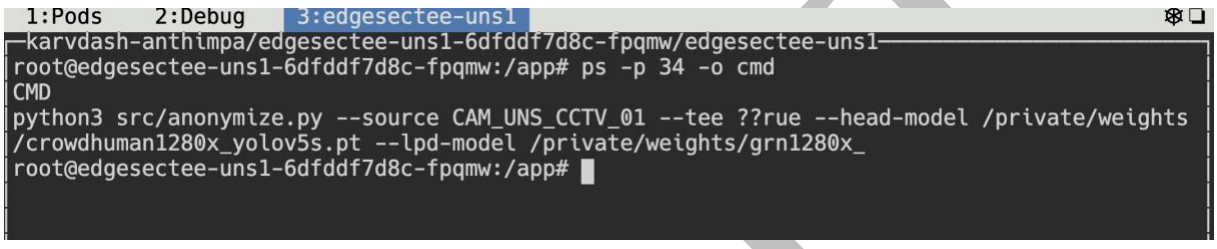

Figure 30. Result of ps command after sconification

Based on the above, we have successfully removed the sensitive information of the camera feed from the VideoAnony application and transferred this information to a Trusted Execution Environment where the Flask application resides.

3.5 KPIs

In this section, we will present and discuss the Key Performance Indicators (KPIs) associated with EdgeSec TEE, specifically related to Task 4.3.

3.5.1 Project-related KPIs

Although KPIs have been already elaborated and addressed in the previous version of D4.2, we deem it necessary to reiterate them for completeness. Table 8 enumerates the project-related KPIs that concern the component EdgeSec TEE, which is implemented in Task 4.3 in the context of the MARVEL platform.

The goals of the project-related KPI with identifier KPI-O3-E3-1 concerning EdgeSec TEE are achieved by attesting edge devices to ensure that no untrusted components flood the system with fraud data and leveraging in-chip memory encryption technologies that enhance the security of the processing devices. The current version of EdgeSec TEE has been uploaded to the MARVEL registry. At its current state, EdgeSec TEE enables programming using the Python language and Flask-related libraries, supporting the VideoAnony component in the UNS pilot.

Regarding the project-related KPI with the identifier iKPI-2.2, the system is protected from the harmful effects of at least three attacks. As already discussed in Section 2.5.1, FORTH currently provides EdgeSec VPN that secures network communications and EdgeSec TEE that offers trusted and protected execution.

Table 8: Project-related KPIs that concern EdgeSec TEE

KPI ID	KPI Description	Strategy	Related Task	Related Component
KPI-O3-E3-1	Realise a secure computing framework at all the processing layers.	Ensure that there are no weak links in the E2F2C chain, every layer and communication channel between them shall be secure.	**T4.3**	**EdgeSec VPN, EdgeSec TEE**
iKPI-2.2	At least three (3) different cyber threats avoided due to E2F2C.	Cyber threats that could otherwise potentially be exploited to harm the system, but their impact is minimised due to the security features offered by E2F2C.	**T4.3**	**EdgeSec VPN, EdgeSec TEE**

3.5.2 Component-related KPIs

The component-related KPI that concerns EdgeSec TEE is presented in Table 9. EdgeSec TEE contributes to the avoidance of cyber threats, along with EdgeSec VPN. EdgeSec TEE utilises the robust Intel SGX technology to provide a comprehensive set of security and privacy assurances in untrusted environments. By leveraging Intel SGX, EdgeSec TEE combines cybersecurity measures, encryption techniques, and verification capabilities, offering a powerful combination to safeguard sensitive data and ensure the integrity of the system. As we showed in Figure 21, not only malicious unauthorised users are prohibited from accessing sensitive data but even admins of the host environment are also excluded. Thus, EdgeSec TEE greatly contributes to the realisation of this specific KPI, by effectively avoiding physical and cyber threats.

Table 9: Component-related KPIs that concern EdgeSec TEE

KPI	Metric	Expected Result	Relevant Project KPI
Effectiveness for Avoiding Cyber Threats	Number of threats avoided	3 distinct cyber threats avoided	*KPI-O3-E3-1*, *iKPI-2.2*

4 GPU-accelerated Stream Processing on the Edge (GPURegex)

4.1 Introduction and Objectives

In this section, GPURegex is presented, described, and evaluated as a component that operates as part of MARVEL. GPURegex is the component that brings performance acceleration features to the complete E2F2C framework, developed within the MARVEL project. As already stated in D4.2, GPURegex is based on the Single Instruction – Multiple Data (SIMD) type of parallel processing, taking advantage of modern processors (either CPUs or accelerators). The component is offered as an OpenCL program that can be executed on top of a hardware device when the proper runtime and libraries are installed. GPURegex accelerates the pattern-matching functionality. Specifically, GPURegex aims to accelerate the matching of particular keywords against audio and video captions exported by the AAC component (TAU). Matching keywords indicate discrete events as defined by MT in the context of the second use case, i.e., MT2 - Detecting Criminal and Anti-Social Behaviours. Each event reported by GPURegex produces an alert.

In this section, we outline the contributions of GPURegex in the context of MARVEL, we summarise the state-of-the-art on pattern-matching and stream processing, while we present how GPURegex advances this state-of-the-art. In addition, we describe the deployment and integration details of GPURegex, and how it communicates with the related MARVEL components, while we evaluate the operation of GPURegex in the corresponding use case (i.e., MT2), concerning the related project and component KPIs.

4.1.1 Contributions

In the context of the MARVEL project, FORTH's years of expertise in accelerated pattern-matching tailored for network security applications (e.g., [18],[19],[26],[28],[29],[31]), adopts a new form to advance the processing performance of other applications in smart cities use cases.

GPURegex contributes to MARVEL with the acceleration of criminal and antisocial event detection. More specifically, GPURegex performs fast pattern-matching against captioning data (resulting from the AAC component, provided by TAU) as exported from raw audio/video files captured by microphones and cameras placed in public spaces (e.g., squares).

4.1.2 Summary of the State-of-the-Art

Pattern matching involves determining whether a specific keyword or pattern is present within an expression or input. The computational nature of pattern-matching enables parallelisation, depending on the chosen algorithm, which facilitates streaming capabilities.

Pattern-matching serves as a fundamental operation in various network packet processing applications, including firewalls, intrusion detection, L7 filtering, and traffic classification. While these applications are commonly associated with pattern-matching, it is important to note that pattern-matching also finds utility beyond network processing. It plays a crucial role in diverse domains such as system log processing, continuous monitoring of healthcare, transportation, and manufacturing data, as well as bioinformatics applications like RNA structure alignments. Consequently, there is a wide range of research areas that benefit from optimised versions of traditional pattern-matching algorithms.

In addition, GPUs have gained significant popularity for their ability to greatly enhance performance in various pattern-matching applications, resulting in substantial speed improvements. Examples of related research include GPU-accelerated intrusion detection

[17][18][19][20][23][24], cryptography [21], and IP routing [22], among others [25][26]. Besides discrete GPUs, shared integrated GPUs, packed with the main processor in the same die, have been used to accelerate or offload parallelisable applications (such as network packet processing) [27][28][29][30][31]. Related works have been discussed in deliverable D4.2 in more detail.

4.1.3 Beyond the State-of-the-Art

The GPURegex component utilises the Aho-Corasick algorithm [1], which is a highly popular and effective method for string pattern-matching. Aho-Corasick is particularly well-suited for searching multiple patterns simultaneously, making it the optimal solution in such scenarios. In addition, the processing nature of the Aho-Corasick algorithm enables the parallelisation of the underlying computations, thus the utilisation of hardware accelerators (such as GPUs) enhances the processing performance significantly.

As discussed in Section 4.1.1, FORTH has expertise in the development of fast pattern-matching engines, mostly tailored for network packet processing applications (e.g., [18],[19],[26],[28],[29],[31]). In the context of MARVEL, FORTH develops and evaluates the GPURegex component that is used in a processing pipeline for criminal and antisocial event detection. In this pipeline, GPURegex receives captioning data from AAC, extracted by raw audio/video files from microphones and cameras that are placed in public locations. GPURegex alerts if specific keywords are matched. These keywords signify discrete criminal and antisocial events (e.g., fighting).

4.1.4 Component Modifications after D4.2

After the end of M18 and the publication of the deliverable D4.2, there have been performed several modifications that concern GPURegex. These modifications mostly refer to the deployment, integration, and testing of the component, since the implementation of the tool remains in the same state.

In D4.2, GPURegex was only deployed at FORTH premises. Currently, GPURegex is deployed on the MTFOG2 machine that is located on the premises of FBK. As exposed by the machine naming scheme, GPURegex participates in the fog layer of MARVEL and operates for the MT pilot and specifically the MT2 use case “Detecting Criminal and Anti-Social Behaviours”. In Section 4.2.2, we present the deployment information in detail.

In addition, in the first version of this deliverable (D4.2), we did not elaborate on the integration details of the GPURegex component, since the AAC component, which provides input to GPURegex, was expected to be deployed after M18. Currently, GPURegex has been integrated with MARVEL and now it is able to communicate with all the related components that participate in the same use case (i.e., AAC, DatAna, and SmartViz). In Section 4.2.3, we discuss the details of the integration.

Finally, in this version of the deliverable, we present the experimental results of the GPURegex component using as input, the output resulting from the AAC component. Specifically, we demonstrate the testbed setup and the experiments that were performed, and we present the final results in Section 4.4.

4.2 Deployment and Integration

GPURegex is a pattern-matching engine that operates in real-time and achieves high-speed performance by harnessing the parallel computing capabilities of general-purpose GPUs (GPGPUs). Its purpose is to accelerate the process of matching strings and/or regular expressions. In MARVEL, GPURegex is utilised for the acceleration of the pattern-matching

procedure against captioning data exported from the AAC component. When GPURegex locates specific keywords against the input, it generates alerts that can indicate possible events according to the use case (e.g., anti-social behaviour).

The final version of GPURegex has been uploaded on the MARVEL Docker image registry and can be downloaded or it can be shared upon request. Specifically, GPURegex is offered through three different Docker images that correspond to three different hardware setups: (i) a hardware setup with an Intel CPU[6], (ii) a hardware setup with an Intel CPU and an integrated (on chip) GPU[7], and finally, (iii) a hardware setup with a discrete NVIDIA GPU[8]. In the first hardware setup, the drivers that are installed in the container concern OpenCL-enabled CPUs, and in the second hardware setup, the drivers that are installed in the container are destined for shared, integrated and OpenCL-enabled GPUs, like Intel HD Graphics GPU series. Finally, in the third hardware setup, the drivers that are installed in the container are tailored for dedicated NVIDIA GPUS.

In this section, we present the most recent deployment and integration details for the GPURegex component with respect to the whole MARVEL platform. Since the implementation details of the GPURegex component remain in the same state as in D4.2, we encourage the reader to refer to the deliverable "D4.2: Security assurance and acceleration in E2F2C framework – initial version"[9].

4.2.1 Implementation

As already mentioned, for the implementation details of the GPURegex component, we encourage the reader to refer to the deliverable D4.2.

4.2.2 Deployment

GPURegex can be deployed to any OpenCL-enabled processor or hardware accelerator, such as dedicated, discrete GPUs or shared, integrated GPUs as long as the necessary OpenCL drivers are properly installed. In the first version of this deliverable, FORTH provided (i) one GPURegex container, targeting CPU-only hardware setups that do not contain any GPU (dedicated or shared) and (ii) a second GPURegex container, destined for hardware setups that contain one main processor packed with a shared GPU. In the final version of the deliverable, FORTH introduces a new GPURegex container designed for a hardware setup that, apart from the main processor, it contains a discrete, dedicated GPU, as well. The template that was used for the deployment configuration of the newly added image is displayed in Listing 14.

```
# gpuregex-priviledged-MTFOG2.template.yaml
apiVersion: v1
kind: Service
metadata:
  name: $NAME
spec:
  type: ClusterIP
  ports:
  - port: 8080
  selector:
    app: $NAME
---
apiVersion: apps/v1
kind: Deployment
metadata:
```

[6] https://marvel-platform.eu/image/gpuregex-intel-cpu (tag: 1)

[7] https://marvel-platform.eu/image/gpuregex-intel-gpu (tag: 2)

[8] https://marvel-platform.eu/image/gpuregex-intel-gpu (tag: 4)

[9] https://zenodo.org/record/6821254#.Ys05GS8Rpqs

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

The three Docker images of GPURegex can be downloaded by the MARVEL image registry as demonstrated in Listing 15 (also presented in D4.2) or by downloading the desired images from the MARVEL platform[10].

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

docker pull registry.marvel-platform.eu/gpuregex-intel-cpu:1 (docker container for Intel CPUs only)
docker pull registry.marvel-platform.eu/gpuregex-intel-gpu:3 (docker container for an Intel CPU with a shared Intel GPU)
docker pull registry.marvel-platform.eu/gpuregex-intel-gpu:4 (docker container for an Intel CPU with a discrete NVIDIA GPU)

docker run -it registry.marvel-platform.eu/gpuregex-intel-cpu:1 /bin/sh
docker run -it registry.marvel-platform.eu/gpuregex-intel-gpu:3 /bin/sh
docker run -it registry.marvel-platform.eu/gpuregex-intel-gpu:4 /bin/sh
```

Listing 15: GPURegex container creation commands

As GPURegex is fully deployed and integrated with the MARVEL platform, the component can be also accessed via the Kubebox service[11]. Kubebox is a terminal and web console for

[10] https://marvel-platform.eu/images

[11] https://kubebox-$username$.marvel-platform.eu/

Kubernetes. An example screenshot is shown in Figure 31. As shown in the screenshot, the GPU that is contained in the MTFOG2 machine is a NVIDIA Tesla K40c card, along with an Intel Xeon CPU. The command "clinfo -l" shows all the devices of a system, with the appropriate OpenCL drivers installed. Each platform corresponds to a distinct hardware vendor (in our example the two vendors are Intel and NVIDIA).

Figure 31. A screenshot of the GPURegex Docker container operating on MTFOG2, accessed via the Kubebox service

In this container, we can execute GPURegex either on the CPU or the GPU. Figure 32 displays GPURegex running on the NVIDIA GPU. In the first console, we can see the output of GPURegex (i.e., total matches and execution time). In the second console, the output of nvtop[12] is shown. The utility nvtop enables the real-time monitoring of tasks executing on the GPU, like (h)top[13] enables CPU task monitoring. Through nvtop we can see several information about the task that is executing, such as the command that issued the execution, CPU and host memory statistics, GPU core and memory utilisation, total GPU temperature and energy consumption.

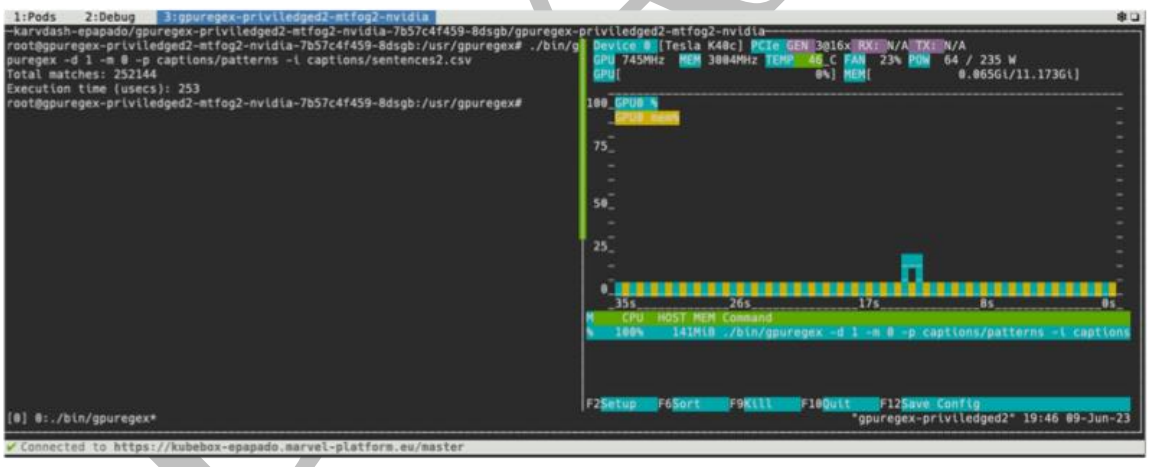

Figure 32. GPURegex running on MTFOG2

4.2.3 Integration with MARVEL

The integration of GPURegex into the MARVEL platform enables its communication with the AAC and SmartViz components. The interconnection of the three components is accomplished via DatAna, operating on the fog layer. The output of the AAC component (provided by TAU) is available to GPURegex through the DatAna instance (MQTT broker: mqtt-mtfog2.karvdash-datanamtfog.svc, topic "AAC"). GPURegex, then, operates and reports any keyword matches against this input that could signify a specific event. In the MT2 use case that GPURegex

[12] https://manpages.ubuntu.com/manpages/focal/man1/nvtop.1.html

[13] https://manpages.ubuntu.com/manpages/xenial/man1/top.1.html

participates, a single event corresponds to a criminal or anti-social behaviour. Each GPURegex match report is published to the same MQTT broker (topic "GPURegex") that is consumed and visualised by SmartViz as an alert. The data model of GPURegex is presented in Listing 16, while an example payload is displayed in Listing 17.

List of properties defined as output

- **startTime**: data type: float (seconds), absolute timestamp for the event start. Following the ISO8601 UTC format.
- **endTime**: data type: float (seconds), absolute timestamp for the event end. Following the ISO8601 UTC format.
- **cameraId**: Unique identifier of the camera from where the video was taken.
- **dateProcessed**: Timestamp of the processing of the entity by the inference model.
- **id**: Unique identifier of the entity. Formatted according to the NGSI standard.
- **MLModelId**: The id of the MLModel that generated this event.
- **owner**: A field containing a JSON encoded sequence of characters referencing the unique Ids of the owner(s). In the case of MARVEL, we can use the pilot's name and use case here.
- **type**: The type of the entity. In this case "Alert".
- **detectedBy**: Link to the device providing where the alert is detected (the link to the device id of the infrastructure in MARVEL).
- **alertSource**: Source of the alert (e.g., "GPURegex_MT_v01").
- **category**: Category of the Alert.
- **subCategory**: Describe the sub category of alert.

Listing 16: The Data Model for the GPURegex output of the MARVEL framework


```
{
  "owner": "MT2",
  "startTime": 2022-04-11T06:40:10.063Z,
  "endTime":2022-04-11T06:40:14.063Z,
  "id": "GPUREGEX_2022-04-11T06:40:10.063Z",
  "type": "Alert",
  "cameraId": "Cam-MGARR",
  "dateProcessed": "2022-04-11T06:40:10.063Z",
  "MLModelId": "GPURegex_MT_v01",
  "detectedBy": "MTFOG_2",
  "alertSource": "GPURegex_MT_v01",
  "category": "traffic",
  "subCategory": "carAccident"
}
```


Listing 17: Example payload in JSON format

4.3 Use Cases and Related Components

In this section, we present the MARVEL components that are paired with GPURegex in a processing pipeline as presented in Figure 33. These components are AAC, SmartViz and DatAna, and they are briefly described in Section 4.3.1. Although AAC is described in the initial version of this deliverable, we also include it in this final version as well, for the purpose

of completeness. In addition, we include the description of the SmartViz component, which is responsible of the visualisation of the GPURegex output through Alerts. Finally, we briefly describe the DatAna component, since it facilitates the intercommunication of GPURegex with AAC and SmartViz. Then, in Section 4.3.2 we describe the MT2 use case, in which GPURegex participates.

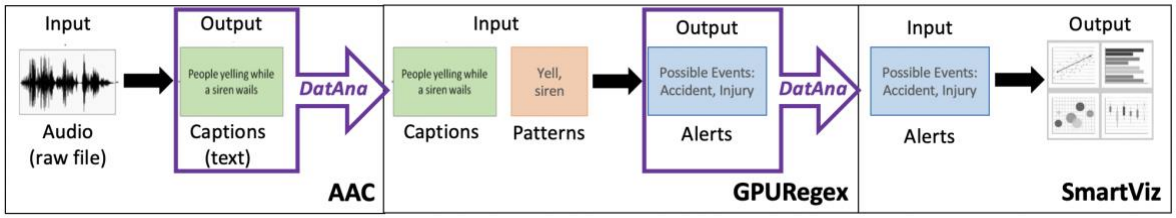

Figure 33. Processing pipeline of AAC, GPURegex, SmartViz and DatAna

4.3.1 Related Components

Automated Audio Captioning (AAC)

AAC is a cross-modal translation task in artificial intelligence that connects audio processing and natural language processing. The automated audio captioning component generates a descriptive textual description to describe the content of an audio clip. These textual descriptions can be used to assist the decision-making process in other components. The AAC component can collaborate and enhance the predictive behaviour of other components by providing high-level information about the audio content.

The input to the AAC component is an audio signal, and the output is a sentence that describes the audio signal, for instance, “Birds chirp while people talk in the background”. The output is a properly formulated sentence that not only describes the sound events but also the spatial-temporal relation between different objects as well as the activities involved. In the learning stage, the component learns the mapping between the input audio signal and the corresponding captions, and at the inference stage, the component predicts the captions for audio input. The learned knowledge depends on the data utilised during the learning stage. The system development requires an audio dataset consisting of audio samples and the corresponding manually generated reference captions. The current state-of-the-art for the AAC system is based on relatively large neural networks with encoder-decoder architectures or transformer-based sequence-to-sequence architectures.

DatAna

DatAna is an Apache NiFi ecosystem-based tool that allows the creation of almost zero-code data flows for ingestion, transformation, enrichment and movement across the computing continuum. DatAna has been described in more detail in the scope of deliverables D2.4[14] and D5.6[15]. In relation to GPURegex, as in the case of any other inference component, the operative is that the component output their results to a specific MQTT broker queue (named as the component: GPUREGEX) available in the same node (edge or fog) where GPURegex service is executing. DatAna subscribes to this queue and proceeds to transform the outputs to the available data models agreed in MARVEL for Alert, Anomaly or MediaEvent and transmit the data to the DFB.

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

[15] MARVEL D5.6 - MARVEL Integrated framework – final version, 2023. To appear.

SmartViz

SmartViz is a versatile data visualisation toolkit that gives the opportunity to 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 is capable of connecting with multiple data sources and then uses its internal data API and configuration options to produce predefined as well as user-defined visualisation dashboards. For the MT2 use case, the Alerts widget has been integrated into SmartViz to address the user's requirement of being alerted when an anomaly occurs. Real-time alerts are detected by GPURegex and transmitted to SmartViz through Kafka messages via the DFB component. These alerts are then visualised by SmartViz, enabling users to quickly respond and take appropriate actions based on the detected anomalies.

4.3.2 GPURegex in MT Use Case

GPURegex participates in the use case MT2 "Detecting Criminal and Anti-Social Behaviours". In this specific use case, MARVEL is used to monitor public areas to detect criminal or anti-social behaviours. GPURegex triggers an alert, once such behaviour is identified.

The MARVEL framework is deployed to detect possible dangerous situations, especially during the night time, such as gatherings, robberies, and aggressions. The system has to analyse the visual and audio data streams of the cameras already installed in the selected locations and send an alert to the local police operational centre in order to monitor the location. Moreover, the stream is saved into the local server of the local police for any further investigation.

GPURegex receives as input captioning data resulting from AAC of specific events reported above (gatherings, robberies, aggressions, etc.). MT has generated two datasets "TrentoOutdoor - real recording" and "TrentoOutdoor - staged recording" (as defined in D2.1[16] and D8.2[17]) by processing the microphone streams from Piazza Santa Maria Maggiore. MT, in collaboration with FBK, collected and anonymised 130 raw audio samples, which correspond to a total of 3 hours. The size of the samples is 11 GB in total. The samples have been used to train the AAC component and they contain both anomalous and non-anomalous situations. The files were annotated by 3 different annotators.

Figure 34 displays alerts that originate from GPURegex. In the example demonstrated, GPURegex reports, within MARVEL, the detection of antisocial activity captured on the MT camera (identifier Cam-MT2-AA-01) on a certain timestamp. A list with the latest alerts is presented within red squares (highlighted with a black, dashed rectangle on the left of Figure 34), and the last alert is displayed in the centre of the page within a red square (highlighted with a black, dashed rectangle).

[16] MARVEL D2.1 - Collection and analysis of experimental data, 2021. https://doi.org/10.5281/zenodo.5052713.

[17] MARVEL D8.2 - MARVEL Data Management Plan, 2021. Confidential.

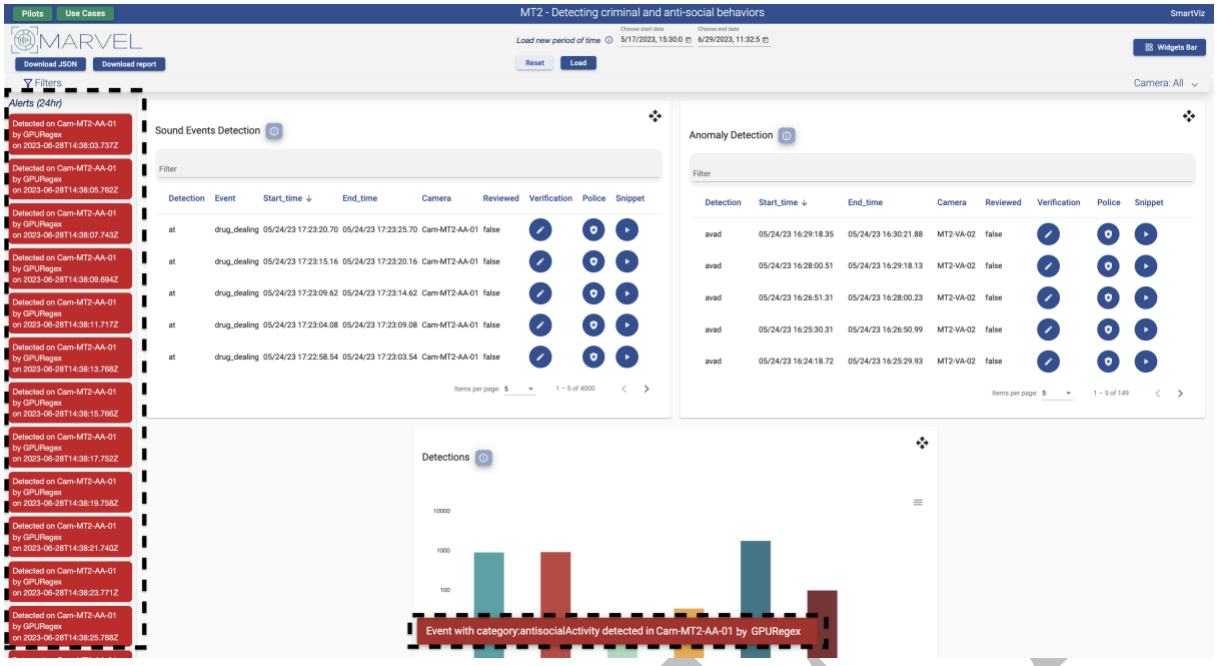

Figure 34. GPURegex alerts in MARVEL

4.4 Experimental Results

In this section, the final experiments and experimental results of the GPURegex component are described.

4.4.1 Testbed Setup

Currently, GPURegex can be executed either on top of an Intel CPU, an integrated Intel GPU, or/and a discrete NVIDIA GPU. This capability is available in the MARVEL image registry via three different Docker images “gpuregex-intel-cpu” (tag:1), “gpuregex-intel-gpu” (tag:3), and “gpuregex-intel-gpu” (tag:4), respectively. Since in the initial version of this deliverable (D4.2) we evaluated GPURegex on the CPU, in this final version of the deliverable we also test GPURegex on top of a discrete NVIDIA GPU.

The machine used is named MTFOG2, provided by FBK for the realisation of the use case. It is equipped with an Intel(R) Xeon(R) E5-1620 CPU operating at 3.60GHz, with a main memory of 19.5GBytes. It contains 4 physical cores (8 logical), with a L1 cache of 512 KB, a L2 cache of 2.0 MB, and a L3 cache of 10 MB. The NVIDIA GPU is a Tesla K40c with 2880 cores and GDDR5 memory with 12GB size.

The Docker version utilised is 20.10.21. The Docker image has a Linux kernel (5.4.0-149-generic) with an Ubuntu 18.04.6 LTS operating system. We have two different OpenCL versions installed; one OpenCL installation for each different device vendor (i.e., Intel CPU and NVIDIA GPU). For the Intel CPU, OpenCL 1.2 is installed, under the platform name “Intel(R) OpenCL” and driver version 1.2.0.475. For the NVIDIA GPU, OpenCL 1.2 is installed as well, this time under the platform name “OpenCL 1.2 CUDA” and driver version 460.106.00.

4.4.2 Experiments

To evaluate the performance of GPURegex, we perform two different sets of experiments.

First, we perform several micro-benchmarks to measure the processing throughput and time achieved using different versions of GPURegex and hardware devices, executing against

diverse input sizes and workloads. Then, using the same input files, we compare the performance sustained with GNU Grep, as we did in D4.2. The input used for testing is provided in JSON format. An example is presented in Listing 18.

{"_index":"aac","_type":"_doc","_id":"s5oQTogBWJo9M5pRZE_q","_score":0.2654129,"_ignored":["dateModified","timestamp"],"_source":{"id":"AACEvent_Cam-MT2-AA-01_2023-05-24T14:01:15.334Z_a_large_yelling_and_a_car_is_driving_by","name":"AAC-2023-05-24T14:01:15.334Z","eventType":"a large yelling and a car is driving by","detectedBy":"MTFOG2","startTime":"2023-05-24T14:01:14.466Z","endTime":"2023-05-24T14:01:19.466Z","dateDetected":"2023-05-24T14:01:15.334Z","owner":"MT2","cameraId":"Cam-MT2-AA-01","MLModelId":"AAC_MT4_v00","type":"MediaEvent","dateStored":"2023-05-24T14:01:17.276Z","dateCreated":"2023-05-24T14:01:16.864Z","timestamp":"","verified":"false","dateProcessed":"2023-05-24T14:01:16.955Z","reviewed":"false","dateModified":"","dateArrived":"2023-05-24T14:01:17.122Z"}}

Listing 18. Example input in json format

For the throughput and execution time benchmarks, we construct and use 9 diverse synthetic input files following the format presented in Listing 18. The characteristics of each input file are shown in Table 10. The results of GPURegex against the different input files are presented in Section 4.4.3.

Table 10: Characteristics of input files used for micro-benchmarks

Input file name	Size in MB	Number of messages included	Workload
AAC_input_nomatch_ssmall	6MB	8,000	0% matches
AAC_input_somematch_ssmall	6MB	8,000	40% matches
AAC_input_fullmatch_ssmall	6MB	8,000	100% matches
AAC_input_nomatch_small	12MB	15,000	0% matches
AAC_input_somematch_small	12MB	15,000	40% matches
AAC_input_fullmatch_small	12MB	15,000	100% matches
AAC_input_nomatch	26MB	33,640	0% matches
AAC_input_somematch	26MB	33,640	40% matches
AAC_input_fullmatch	26MB	33,640	100% matches

For the second experiment, we test the interconnection of GPURegex with DatAna and the relevant MQTT broker operating on MTFOG2. To perform this test, MQTT messages, originating from AAC, are published to the “AAC” topic. GPURegex is subscribed to the “AAC” topic, so it consumes the incoming data in real time. For each incoming AAC MQTT message, GPURegex is invoked and if it detects any of the patterns against the input, GPURegex reports an alert. This alert generates an outgoing MQTT message that is published to the “GPURegex” topic. From this specific topic, SmartViz can consume the alerts reported by GPURegex and visualise them. We perform this experiment using diverse time intervals. We start sending 1 MQTT message per multiple seconds (i.e., 1 MQTT message per 4 seconds) and we achieve rates of 1 MQTT message per multiple subdivisions of 1 second (i.e., 8 MQTT messages per 1 second). Aiming to stress the system, we include input that forces intensive workload conditions (i.e., 100% matches).

To identify the patterns that concern the pilot use case and to construct the automata required for the GPURegex execution, we contacted MT. Since GPURegex participates in the MT2 use case entitled “Detecting Criminal and Anti-Social Behaviours”, we decided to include patterns that indicate such activities and events. Example patterns that we use are: “fight”, “attack”, “scream”, and “yelling”.

4.4.3 Final Results

As described in Section 4.4.2, we evaluate GPURegex using two sets of experiments. Table 11 and Table 12 present the performance achieved from the execution time and processing throughput micro-benchmarks (mean values of 30 consecutive runs).

Specifically, Table 11 shows the execution time measured in microseconds (μs). In Table 12, the processing throughput of GPURegex is displayed, measured in Mbps (Mbits/second). In the column "GNU Grep", we present the performance of the GNU grep utility that is also based on the Aho-Corasick algorithm (using the -F option that enables fixed strings searching) and runs on the CPU. In the column "GPURegex (CPU)", we present the performance of GPURegex running on the CPU (Intel(R) Xeon(R) E5-1620), while in the column "GPURegex (CPU-opt/ed)", we present the performance of an optimised version of GPURegex, tailored for CPUs and shared GPUs. This optimised version of GPURegex enables memory mapping that significantly improves the processing performance. Finally, the column "GPURegex (GPU)" displays the performance of GPURegex running on the NVIDIA Tesla GPU.

Based on the performance achieved, we can conclude that GPURegex always outperforms the GNU Grep utility with an x3.5 speedup, at minimum. The best processing performance is achieved by GPURegex on top of the Intel Xeon CPU when the memory optimisation is enabled. Comparable results are also achieved by the dedicated NVIDIA GPU, which is also optimised to minimise memory copies through the PCIe bus (to/from the host memory).

Table 11: Processing time of GPURegex and GNU Grep (measured in microseconds)

Input Size	Matches	GNU Grep utility (CPU only)	GPURegex (CPU)	GPURegex (CPU-opt/ed)	GPURegex (GPU)
6MB	0%	19,000 μs	4,930 μs	2,160 μs	2,630 μs
6MB	30%	16,000 μs	4,880 μs	2,120 μs	2,590 μs
6MB	100%	14,000 μs	4,850 μs	2,250 μs	2,570 μs
12MB	0%	29,000 μs	4,100 μs	1,680 μs	1,850 μs
12MB	30%	24,000 μs	4,120 μs	1,720 μs	1,820 μs
12MB	100%	18,000 μs	4,300 μs	1,830 μs	1,810 μs
26MB	0%	47,000 μs	670 μs	310 μs	510 μs
26MB	30%	43,000 μs	670 μs	340 μs	520 μs
26MB	100%	27,000 μs	720 μs	350 μs	530 μs

Table 12: Processing throughput of GPURegex and GNU Grep (measured in Mbits/second)

Input Size	Matches	GNU Grep utility (CPU only)	GPURegex (CPU)	GPURegex (CPU-opt/ed)	GPURegex (GPU)
6MB	0%	2,526 Mbps	9,736 Mbps	22,222 Mbps	18,250 Mbps
6MB	30%	3,000 Mbps	9,836 Mbps	22,642 Mbps	18,532 Mbps
6MB	100%	3,428 Mbps	9,897 Mbps	21,333 Mbps	18,677 Mbps
12MB	0%	3,310 Mbps	23,415 Mbps	57,143 Mbps	51,892 Mbps
12MB	30%	4,000 Mbps	23,301 Mbps	55,814 Mbps	52,747 Mbps
12MB	100%	5,333 Mbps	22,326 Mbps	52,459 Mbps	53,039 Mbps
26MB	0%	4,425 Mbps	310,447 Mbps	670,967 Mbps	407,843 Mbps
26MB	30%	4,837 Mbps	310,447 Mbps	611,764 Mbps	400,000 Mbps
26MB	100%	7,703 Mbps	288,889 Mbps	594,286 Mbps	392,453 Mbps

Concerning the second experiment, MQTT messages originating from AAC are published to the "AAC" topic. GPURegex consumes them and reports any alerts to the "GPURegex" topic. The goal of this experiment is to show that real-time processing is feasible using the GPURegex component. In Figure 35, Figure 36, and Figure 37, we can see GPURegex being executed on the MTFOG2 machine, while it processes input coming from the AAC component and publishes alerts for SmartViz to visualise. In the three figures, we can see that GPURegex was

able to successfully consume and publish MQTT messages with receiving rates of (i) 1 message per second, (ii) 4 messages per second, and (iii) 8 messages per second.

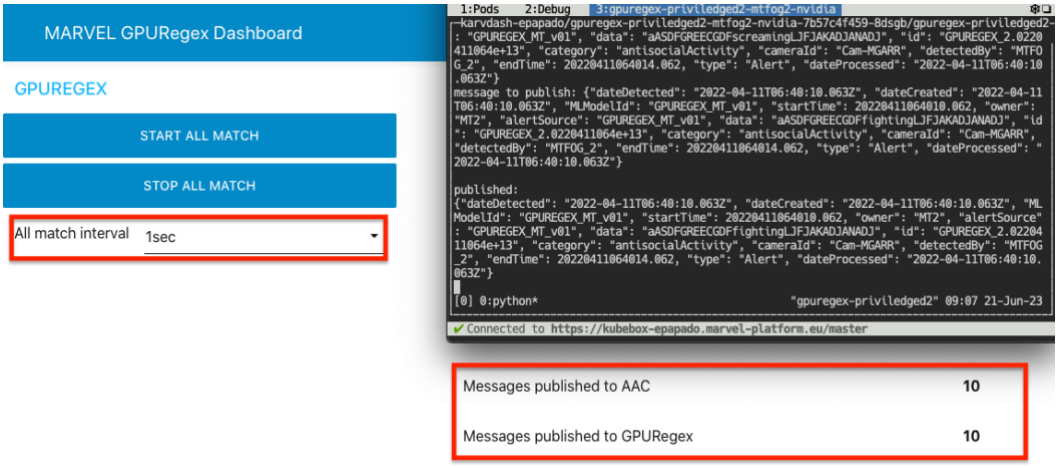

Figure 35. GPURegex consumes from the AAC topic, executes and reports alerts to the GPURegex topic with a rate of 1 MQTT message per second

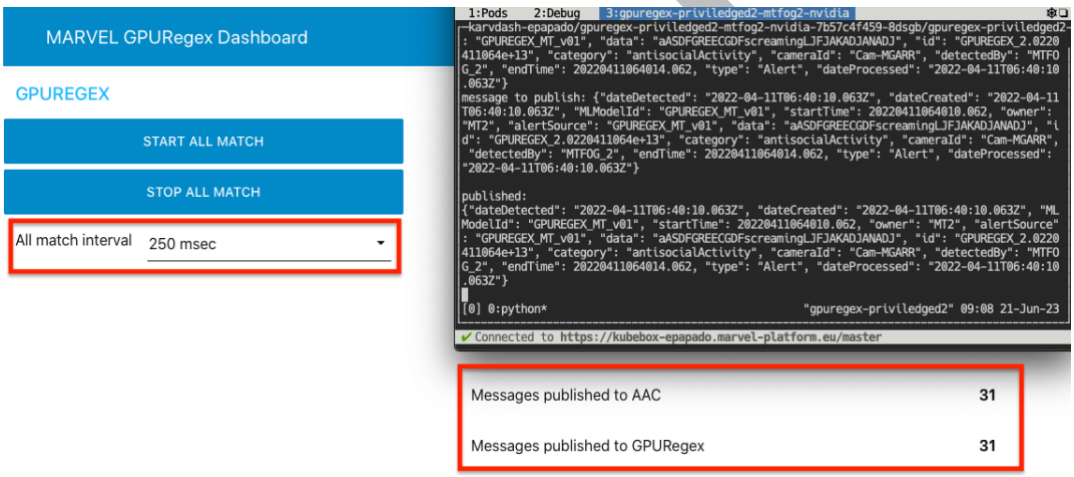

Figure 36. GPURegex consumes from the AAC topic, executes and reports alerts to the GPURegex topic with a rate of 4 MQTT messages per second

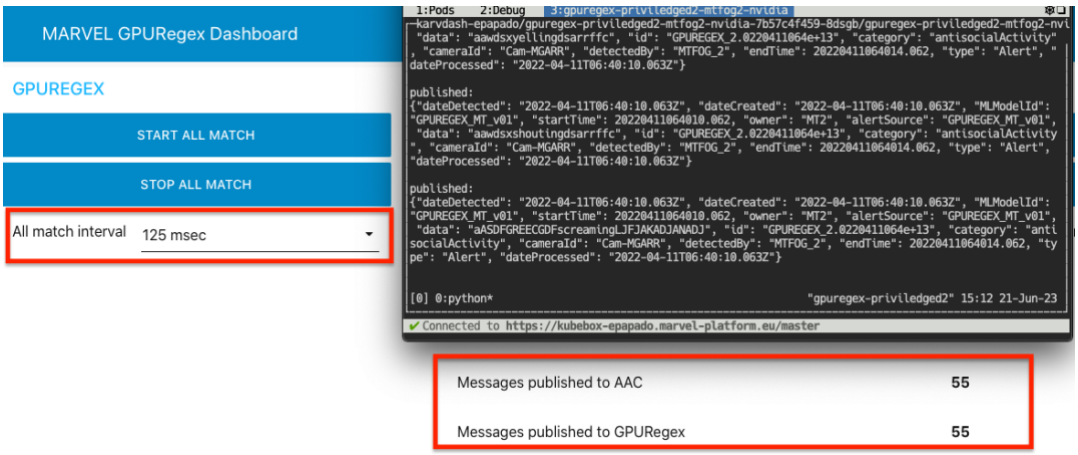

Figure 37. GPURegex consumes from the AAC topic, executes and reports alerts to the GPURegex topic with a rate of 8 MQTT messages per second

4.5 KPIs

In this section, the relation of the GPURegex component to the project- and component-related KPIs in the context of the MARVEL project is presented.

4.5.1 Project-related KPIs

The project-related KPI that concerns the component, namely GPURegex, which is implemented in Task 4.3 in the context of the MARVEL platform is presented in Table 13. More specifically, the KPI proposes the acceleration of the pattern-matching procedure after the utilisation of the GPURegex component. The processing performance is calculated with the performance metrics of throughput and latency. In D4.2, we discussed how we reached *KPI-O1-E1-2* by evaluating the processing performance of GPURegex using three publicly available datasets (by that time, the output of AAC was expected only after M18) and several micro-benchmarks.

In this deliverable, we use the output of AAC for the evaluation of GPURegex processing performance. As shown in Section 4.4, the GPURegex component can accomplish the realisation of *KPI-O1-E1-2*, since not only it achieves a throughput increase and latency decrease by 10%, but it offers at least an x3.5 speedup, when compared to a typical tool with similar functionality (i.e., pattern-matching).

Table 13: Project-related KPIs that concern GPURegex

KPI ID	KPI Description	Strategy	Related Task	Related Component
KPI-O1-E1-2	Increase of data throughput and decrease of access latency by 10%.	Access latency is defined by the response time of the overall system, while the throughput is defined as the amount of data that can be processed per unit of time.	**T4.1, T4.2**	**GPURegex**

4.5.2 Component-related KPIs

The component-related KPI that concerns GPURegex is presented in Table 14. In D4.2, we evaluated GPURegex using the “throughput” and “execution time” metrics (i.e., latency) and three publicly available datasets, by performing several micro-benchmarks. The results of

GPURegex achieved and succeeded the expected processing speed-up, as presented in Section 4.4.3 and discussed in Section 4.5.1.

Table 14: Component-related KPIs that concern GPURegex

KPI	Metric	Expected Result	Relevant Project KPI
Efficiency	Throughput and Execution time	At least 10% processing speed-up	*KPI-O1-E1-2*

5 Conclusions

In this deliverable, entitled D4.5 “Security assurance and acceleration in the E2F2C framework – final version”, we presented the work performed in the context of Task 4.3 “Security and acceleration in the complete E2F2C”, within the scope of WP4 “MARVEL E2F2C distributed ubiquitous computing framework” and the MARVEL project under Grant Agreement No. 957337.

The main participating components, which have been developed in the context of Task 4.3, are presented and discussed. These three components offer security and acceleration features in the complete E2F2C2 MARVEL framework. The security-related components are EdgeSec VPN and EdgeSec TEE. EdgeSec VPN secures the communications using end-to-end network encryption, while EdgeSec TEE shields the execution of sensitive data processing applications within trusted regions of memory. The component that offers acceleration in the pattern-matching procedure is GPURegex, which takes advantage of the SIMD parallel processing architectural design and modern processors, like powerful multi-core CPUs or GPUs.

This deliverable demonstrates the work conducted during the period M19—M30 in Task 4.3 of the MARVEL project. In this deliverable, we present the final versions of the components EdgeSec VPN, EdgeSec TEE, and GPURegex and their results in the use cases in that they participate. Finally, we show how each component achieves the relevant KPIs.

During the final 6 months of the project, the evaluation activities of the MARVEL framework will take place. This phase will involve further refinement of the components based on feedback received from the pilots and external evaluators.

6 References

[1] Hopkins, J. and Green, M. (2019). OpenVPN 2.4 Evaluation Summary and Report. [online] Private Internet Access Blog. Available at: https://www.privateInternetaccess.com/blog/2017/05/openvpn-2-4-evaluationsummary-report/ [Accessed 16 Aug. 2019].

[2] Donenfeld, J. (2018). WireGuard: Next Generation Kernel Network Tunnel [Ebook] (1st ed.). Retrieved from http://www.wireguard.com/papers/wireguard .pdf.

[3] Cisco. (2019). Security and VPN - Support Documentation. Retrieved from https://www.cisco.com/c/en/us/tech/security-vpn/index.html

[4] Ferguson, N., & Schneier, B. (2003). A Cryptographic Evaluation of IPSec. Retrieved from https://www.schneier.com/academic/paperfiles/paper-IPSec.pdf

[5] Schuster, Felix, et al. "VC3: Trustworthy data analytics in the cloud using SGX." *2015 IEEE symposium on security and privacy*. IEEE, 2015.

[6] Zheng, Wenting, et al. "Opaque: An oblivious and encrypted distributed analytics platform." *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. 2017.

[7] Goltzsche, David, et al. "Endbox: Scalable middlebox functions using client-side trusted execution." *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2018.

[8] Trach, Bohdan, et al. "Shieldbox: Secure middleboxes using shielded execution." *Proceedings of the Symposium on SDN Research*. 2018.

[9] Poddar, Rishabh, et al. "{SafeBricks}: Shielding Network Functions in the Cloud." *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*. 2018.

[10] Shinde, Shweta, et al. "Panoply: Low-TCB Linux Applications With SGX Enclaves." *NDSS*. 2017.

[11] Tian, Hongliang, et al. "Sgxkernel: A library operating system optimized for intel SGX." *Proceedings of the Computing Frontiers Conference*. 2017.

[12] Tsai, Chia-Che, Donald E. Porter, and Mona Vij. "{Graphene-SGX}: A Practical Library {OS} for Unmodified Applications on {SGX}." *2017 USENIX Annual Technical Conference (USENIX ATC 17)*. 2017.

[13] Arnautov, Sergei, et al. "{SCONE}: Secure linux containers with intel {SGX}." *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. 2016.

[14] Baumann, Andrew, Marcus Peinado, and Galen Hunt. "Shielding applications from an untrusted cloud with haven." *ACM Transactions on Computer Systems (TOCS)* 33.3 (2015): 1-26.

[15] Kunkel, Roland, et al. "Tensorscone: A secure tensorflow framework using intel sgx." *arXiv preprint arXiv:1902.04413*(2019).

[16] Aho, Alfred V., and Margaret J. Corasick. "Efficient string matching: an aid to bibliographic search." *Communications of the ACM* 18.6 (1975): 333-340.

[17] Smith, Randy, et al. "Evaluating GPUs for network packet signature matching." *2009 IEEE International Symposium on Performance Analysis of Systems and Software*. IEEE, 2009.

[18] Vasiliadis, Giorgos, et al. "Gnort: High performance network intrusion detection using graphics processors." *International workshop on recent advances in intrusion detection*. Springer, Berlin, Heidelberg, 2008.

[19] Vasiliadis, Giorgos, et al. "Regular expression matching on graphics hardware for intrusion detection." *International Workshop on Recent Advances in Intrusion Detection*. Springer, Berlin, Heidelberg, 2009.

[20] Vasiliadis, Giorgos, Michalis Polychronakis, and Sotiris Ioannidis. "MIDeA: a multi-parallel intrusion detection architecture." *Proceedings of the 18th ACM conference on Computer and communications security*. 2011.

[21] Harrison, Owen, and John Waldron. "Practical Symmetric Key Cryptography on Modern Graphics Hardware." *USENIX Security Symposium*. Vol. 195. 2008.

[22] Han, Sangjin, et al. "PacketShader: a GPU-accelerated software router." *ACM SIGCOMM Computer Communication Review* 40.4 (2010): 195-206.

[23] Jamshed, Muhammad Asim, et al. "Kargus: a highly-scalable software-based intrusion detection system." *Proceedings of the 2012 ACM conference on Computer and communications security*. 2012.

[24] Choi, Byungkwon, et al. "{DFC}: Accelerating string pattern matching for network applications." *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*. 2016.

[25] Sun, Weibin, and Robert Ricci. "Fast and flexible: Parallel packet processing with GPUs and click." *Architectures for Networking and Communications Systems*. IEEE, 2013.

[26] Vasiliadis, Giorgos, et al. "{GASPP}: A {GPU-Accelerated} Stateful Packet Processing Framework." *2014 USENIX Annual Technical Conference (USENIX ATC 14)*. 2014.

[27] Go, Younghwan, et al. "{APUNet}: Revitalizing {GPU} as Packet Processing Accelerator." *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. 2017.

[28] Papadogiannaki, Eva, et al. "Efficient software packet processing on heterogeneous and asymmetric hardware architectures." *IEEE/ACM Transactions on Networking* 25.3 (2017): 1593-1606.

[29] Giakoumakis, Giannis, et al. "Pythia: Scheduling of concurrent network packet processing applications on heterogeneous devices." *2020 6th IEEE Conference on Network Softwarization (NetSoft)*. IEEE, 2020.

[30] Kim, Joongi, et al. "NBA (network balancing act) a high-performance packet processing framework for heterogeneous processors." *Proceedings of the Tenth European Conference on Computer Systems*. 2015.

[31] Papadogiannaki, Eva, and Sotiris Ioannidis. "Acceleration of intrusion detection in encrypted network traffic using heterogeneous hardware." *Sensors* 21.4 (2021): 1140.