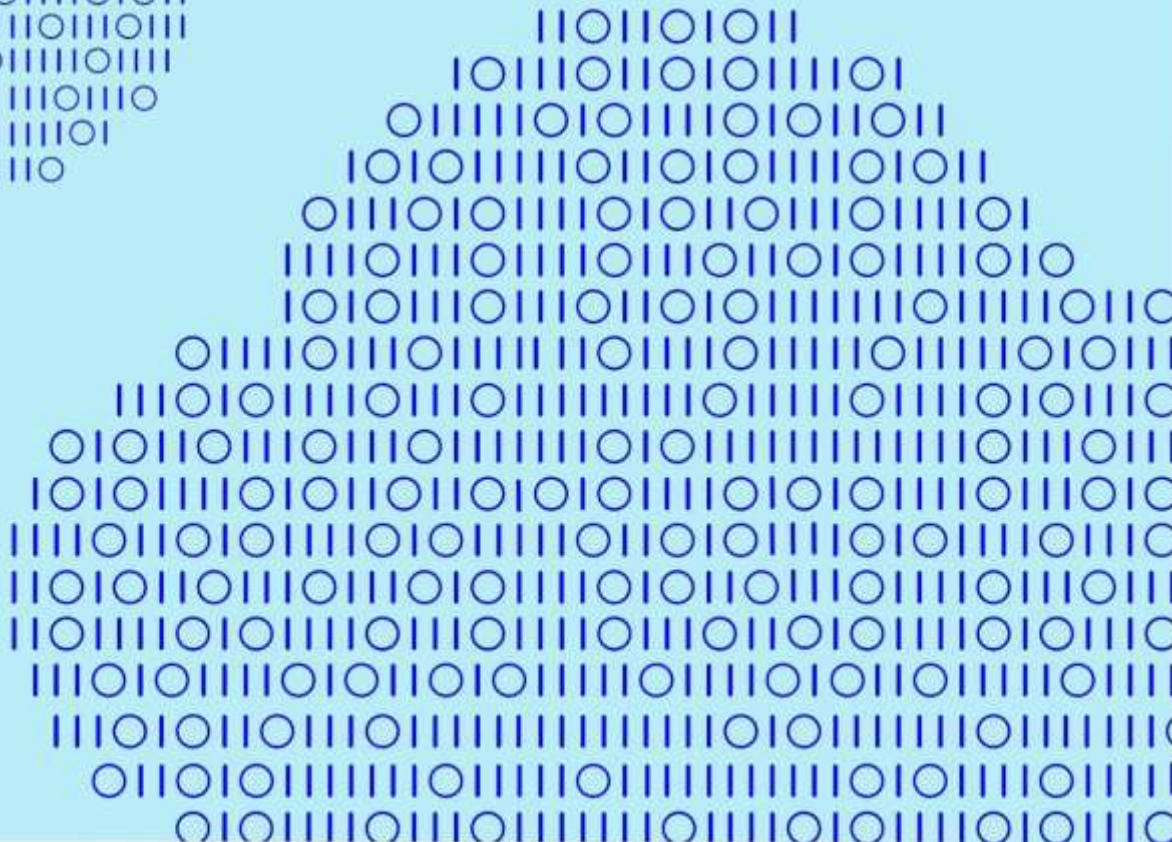
AIDA

# Adaptive, Intelligent and Distributed Assurance Platform



# Deliverable 5.1

## Plan for Software Integration, Validation, and Pilot Activities

**Date** 28/02/2022

**Activity N** 5

**Version** 1.0

**Authors** Pedro Pires, Pedro Fidalgo, Bruno Lages, Pedro Santos, Nuno Aguiar

**Reviewers** Pedro Fidalgo, Carlos Martins

# Table of Contents

# Executive Summary

The AIDA: Adaptive, Intelligent and Distributed Assurance Platform project main goal is the conception of RAID's platform new version having parts of the pipeline moved to the edge of the system. Now, the platform is built using physically co-located servers, either on premises or in the cloud. AIDA intends to provide adaptable and configurable data collection and monitoring while preserving real-time, security and dependability guarantees. Current market demand requires RAID to become more effective in scaling to unprecedented levels while assuring data privacy and confidentiality of the data processed across the different administrative domains and owners. This poses significant and interesting research and development challenges in security of data and infrastructure.

Pilot projects and trials are a good way to reduce risk on projects. The pilot project is an initial small-scale implementation that is used to prove the viability of a project idea. This could involve either the exploration of a novel new approach or idea or the application of a standard approach recommended by outside parties but which is new to the organization. An example of this would be the standard implementation approach for a new off-the-shelf package.

The pilot project enables an organization to manage the risk of a new idea and identify any deficiencies before substantial resources are committed. The pilot can confirm viability and scalability and enable proposed components and procedures to be tested. It will confirm the appropriateness and safety of the tools proposed and security of the platform. It also enables the benefits to be tested and a more reliable investment appraisal to be created for the main project.

# 1. Introduction

RAID is Mobileum's platform that handles the entire risk management lifecycle of enterprises. It comprises a very fluid pipeline, which covers several steps as data collection, monitoring, notification, discovery and actuation, to provide several services. Companies over the world are served by RAID to capture revenue for all services rendered (revenue assurance), for business assurance and fraud management, in an end-to-end real-time manner.

The Adaptive, Intelligent and Distributed Assurance Platform (AIDA) project has the main goal of conceiving a new version of the current RAID platform where some of the pipeline phases can be dynamically moved to the edges of the system. Currently, the platform is fully deployed in physically co-located servers, either on premises or in the cloud. AIDA aims to provide highly configurable data collection and monitoring while preserving real-time, security and dependability guarantees, with ability to run in diverse hardware. The market pressure requires RAID to become effective in scaling to unprecedented levels while assuring data privacy and confidentiality of the data processed across the different administrative domains and owners. This poses significant and interesting research and development challenges in the area of security of data and infrastructure which need to be addressed.

AIDA faces the identified challenges through several perspectives that compose the activities within this project. Namely, focusing on distribution of the platform to encompass the cloud and the edge, which allows the platform to take advantage of 5G networks to have more effective scaling; utilization of distributed and federated machine learning to enable model training on distributed data and to learn with both behavior patterns and context evolution, leveraging this process for risk management purposes; and also assure security of the platform, through intrusion detection and tolerance to assure the continuity of service, and data privacy and confidentiality, which are fundamental concerns in the design of the platform, as it should be able to work with anonymized data, as well as provide secure data exchange between the different processing elements in distinct environments.

This document serves the purpose of defining the baseline for the implementation of a pilot project. Participants from the remaining activities will be able to understand what and how the pilot should be implemented. Defines the uses cases needed for pilot proposes and what frameworks can be used to execute the pilot correctly. Also defines what tests are needed to be executed to achieve a good quality assurance metric.

## 1.1 Target Audience

This document is intended for internal use.
Participants from the remaining activities will be able to understand what and how the pilot should be implemented.

## 1.2 Pilot Definition Overview

To assure a complete pilot for the project, this document covers an approach that covers the relevant topics of the platform. Platform components cannot be devised in an isolated manner as to focus on specific parts of the system but rather as an overall view and definition according to multiple perspectives of the system.

## 1.3 Document Structure

The rest of this document is structured in the following three main sections.
First section is dedicated to describing the main components of the pilot, the solution architecture, Machine learning framework that will be used and how it can be used in order to accomplish the results, the list of uses cases that can be addresses and the security measures applied on data and on the architecture.
The next section describes the implementation plan and the goals achieved through time.
The last section defines the validations that are required to assure that the pilot has the quality assurance acceptance.

# 2. Pilot Definition

AIDA will be tested through a pilot that aims to join all the different tasks coordinated by different consortium partners as specified in activity one. 5G decreases network latency by 10 times when compared to 5G, requiring low latency infrastructure distributed through services that need to be in the edge of the network, with limited and scalable resources, and others with available in the cloud. This requires the scope of activity two, a distributed infrastructure from cloud to edge, which can orchestrate and monitor all activity between the two different infrastructures and the corresponding data lakes.

Since services are distributed between different edges and clouds, and in some cases data needs to comply with law requirements or other constraints which imply limited to none data being transmitted outside the edge, Federated Machine Learning is required in order to join the multiple fraud detection patterns identified in each edge. Activity number three aims to design a framework and trustworthy, self-explainable and privacy preserving anomaly detection algorithms, enhancing federated distributed and Federated Machine Learning.

A key important aspect to ensure the availability and security of the Aida platform is provided by activity four, ensure, a secure communication layer between all network elements, while safeguarding the privacy of the data being processed or stored. This requires security monitoring and actuation in case of intrusions or any other attempts to disrupt the service of the platform or put the data security or privacy of risk.

The pilot definition provides the integration of all these components. The platform will run the algorithms developed in the context of activity three, ensuring that all remain components are successfully integrated, that is, evaluation of the precision of the anomaly detection algorithms, as well as the usability, performance and security of the system.

## 2.1 Architecture Cloud to Edge

To initiate the design of the security strategy, the platform was first analyzed with regard to the scope and concerns that should be addressed by the security and privacy strategy, with the architecture being examined to identify relevant actors and stakeholders as well as possible malicious agents that can endanger the security and privacy of the platform and its data. Figure 1 depicts the general architecture of the AIDA platform.

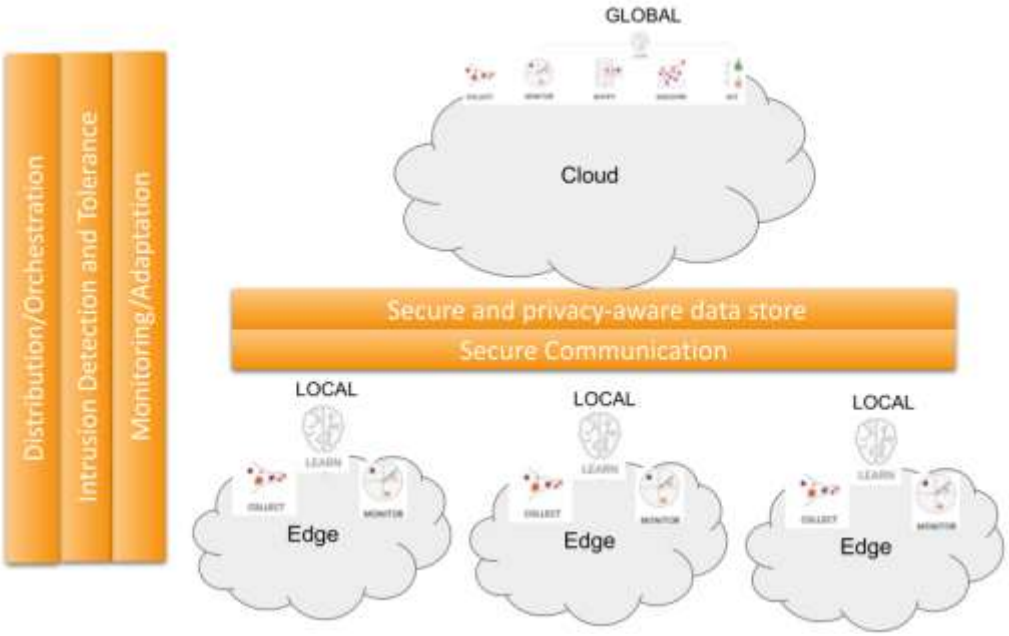



*Figure 1 - AIDA architecture overview*

The AIDA platform describes a distributed architecture with cloud and edge environments for a more global and local processing, respectively. It is composed of three vertical components that motivate environments from centralized cloud towards the edge nodes: comprising the orchestration of the distributed edge nodes for better scalability; monitoring/adaptation of the distributed platform to assure that data is collected for problem identification and for the adaptation of the platform; and intrusion detection and tolerance so that AIDA is resilient to intrusion by detecting and tolerating its occurrence. To complement these components, there are two components that ease the interaction between cloud and the edge to assure that communication channels among services are secured as well as the privacy and security of the data store is also maintained. In more detail, Figure 2.2 depicts the general architecture of AIDA, with the five components previously mentioned and the cloud and edge parts detailed.

The Cloud side of the platform is composed of the main components of the RAID platform divided into the Common, Processing, Data and Presentation areas of the platform. In general, this architecture provides clear distinction between the layers across the cloud deployment and actuation area of each service. The Edge side of the platform is composed of lightweight services delegated to the edge of the network, based on containerized applications deployed to provide real-time, low latency service to the customers while assuring higher and more effective scalability capacity.

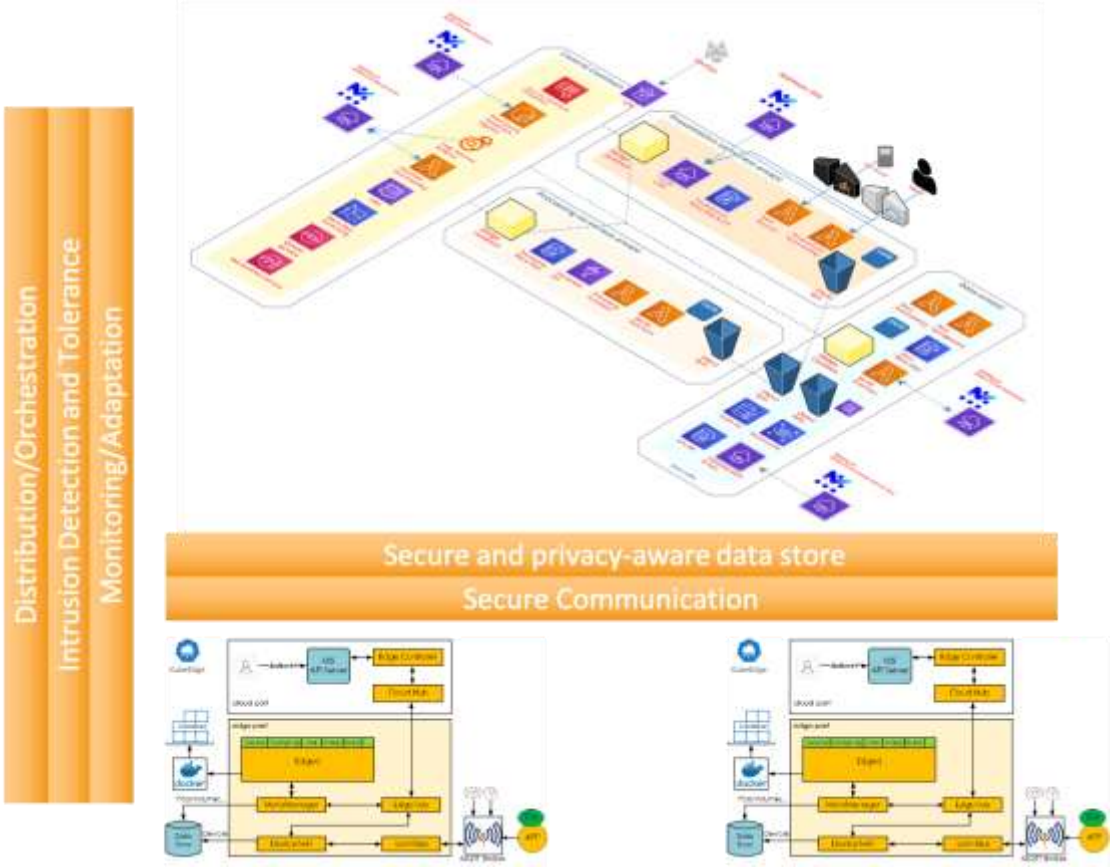



*Figure 2 - AIDA edge and cloud components distribution overview.*

Through these components, the AIDA platform will assure that users are served with an integrated low code platform that allows them to construct several objects (e.g., data flows, case management or views) without having to code. For this, the platform exposes three main components to the exterior, namely the presentation layer, the web services and the Extract, Transform and Load (ETL) layer. The presentation layer serves the users of the platform with user-friendly interfaces, following the Rich Internet Applications (RIA) architecture with state-of-the-art technologies, for the creation, management, and deletion of the objects mentioned above. The users can navigate through the pages available to perform the intended operations. The web services are available to the clients that feed the system with data, communicating through REST and using the Tomcat server as support. With regard to the ETL layer is where the processing of the information collected is conducted which can also partially occur at the edge of the network, as a consequence this layer is also exposed to external agents that can feed the information to get it processed.
In light of this architecture, we distinguish three main sets of stakeholders that are relevant for the analysis.

- **Mobileum**: the company that provides the platform being developed, it is responsible for the management of the services and orchestration of the components, assuring the release of features, maintaining the availability and correct configuration of the application. As a part of this actor, we highlight two important roles: the Developer, responsible for creating objects in the RAID platform; and the Analyst, which uses the objects created to perform intended operations.

- **Service User Entity**: the entity that takes advantage of the service provided by the platform, it relies on it to assure their communications and operations are monitored and secured through the operation of RAID, detection for instance possible fraud committed against the Service User Entity. This can be a Telecom Operator or a company that operates in the context of autonomous driving or Industry 4.0.
- **Data Subjects**: the subscribers or users of the service of the Service User Entity, whose data and communications will be monitored and analyzed through the RAID's platform for the management of operations and detection of possible malicious behavior. These can be the subscribers of a Telecom Operator, the users of autonomous vehicles or plant operators in the context of Industry 4.0.

This was investigated in Task 2.1 where was described in deliverables D2.1 and D2.2. Also Activity 4 has raised complementing aspects to the architecture that impacts in the architecture that are described in deliverable D4.1, D4.2 and D4.4.

This section describes the architecture of the proposed solution presented in this deliverable. Section 2.1.1 presents the architecture of the *Machine Learning* applications. Next, section 2.1.2 describes the implementation in several Kubernetes *Clusters* with use of a *Message Broker* and *Publisher and Subscribe* paradigm. Finally, section 2.1.3 presents the sequence diagrams of the *Machine Learning* applications and the overall scope of the distributed processes.

### 2.1.1 Adopted Technologies

The applications development was mainly based on three technologies – TensorFlow, Flask and Docker – as well as the programming language Python. TensorFlow is one of the most widely used Machine Learning frameworks, focusing on the implementation of Deep Learning algorithms. The choice of TensorFlow was mainly due to the existence of *lite* models with TensorFlow Lite. This module was specifically designed for devices with limited resources such as Edge devices or embedded systems. Moreover, TensorFlow also TensorFlow Federated, a module focused on Machine Learning for decentralized data. TensorFlow Federated was developed for Federated Learning scenarios where a centralized ML model is shared with multiple clients.

Flask is a Python package focused on web application development based on RESTful APIs. It is a relatively simple framework yet still very powerful given its support to external sources such as databases. It is one of the most widely used libraries for creating APIs given its detailed documentation and support of different authentication methods.

Docker is an open-source platform that allows containerization of applications. It allows developers to encapsulate applications in containers: executable components that merge source code with operating system libraries and dependencies and allows to run the code in any given environment. This leads to a simpler delivery process, sharing and distribution of applications and it has become widely popular in software organizations, especially in a multiCloud environment. The choice of

Docker for this project is based on the ease of automation and sharing of the applications created, allowing them to be seamlessly executed and replicated in any given infrastructure.

The Python programming language was used to support the development of the applications, mainly for the development of the Machine Learning (TensorFlow) features and REST APIs (Flask). The choice of this language was based on the fact that is a common denominator to the various libraries used in this project and it has a highly active community.

### • 2.1.2 *Machine Learning* Architecture

This section details the *Machine Learning* architecture developed in the context of this project. The proposed architecture attempts to address the main challenges of the project, such as working on decentralized data and knowledge transfer of local models. This is described in detail next, where the focus is primarily on the global architecture and inner functioning of each *Edge* node and a central node (designated centralized location).

On a typical *Mobile Edge Computing* (MEC) scenario the set of *Edge* nodes do not access the same data set (i.e., each *Edge* node typically has access only to a local data repository). Two main challenges are presented by this architecture that are addressed by the proposed solution. Firstly, each node may not possess enough computational resources in order to train *Machine Learning* models (especially if using *Deep Learning* algorithms) or even to perform inference (predictions). Secondly, given that each Edge node is associated to a local repository, there is no data centralization, as each node might have to use distinct data to train models. Figure 3 depicts the proposed architecture for meeting these *Machine Learning* goals:

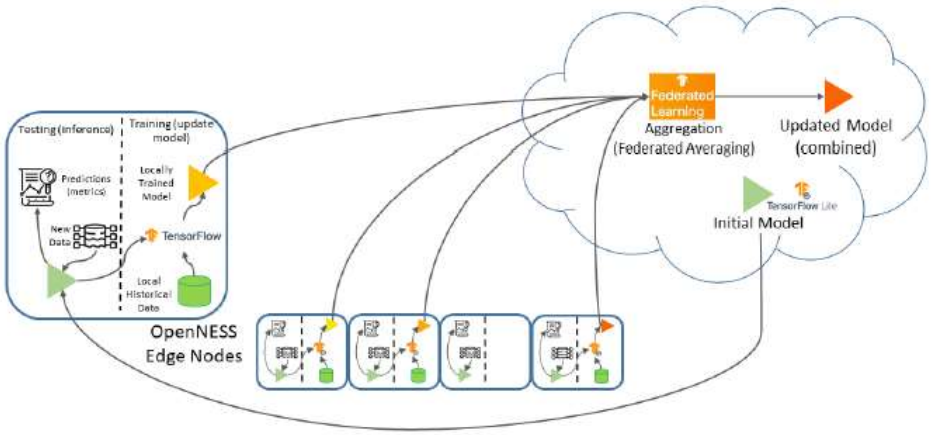



*Figure 3 - Proposed architecture to address the identified Machine Learning challenges.*

### • 2.1.1.1*Edge* Nodes

In this architecture, the flow in each *Edge* node is divided in two main phases:

- Predictions: At any time there is always a common ML model that is shared across all Edge nodes (model copies). The model is in the format *TensorFlow Lite*, a way to decrease both the prediction time and the disk space required. This model performs predictions whenever necessary (in *batch* or streaming/mini *batch*).

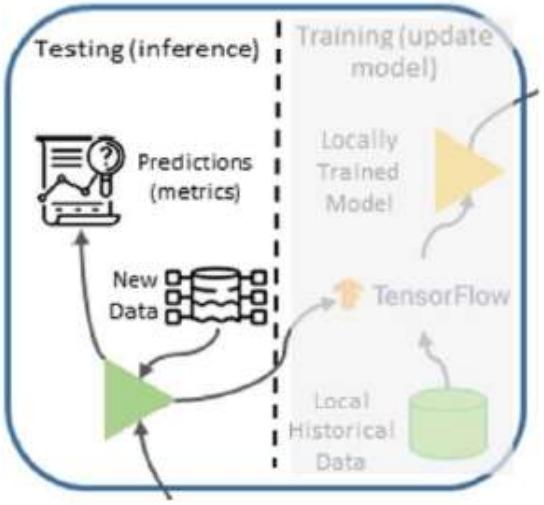



*Figure 4 - Edge node internal workflow (predictions)*

- Training: The training phase consists in the creation of a new model in each Edge node. Each node uses a local set of data to use for training (the local data can be used for more than one Edge node, depending on the case). The training uses the base architecture of the current shared model.

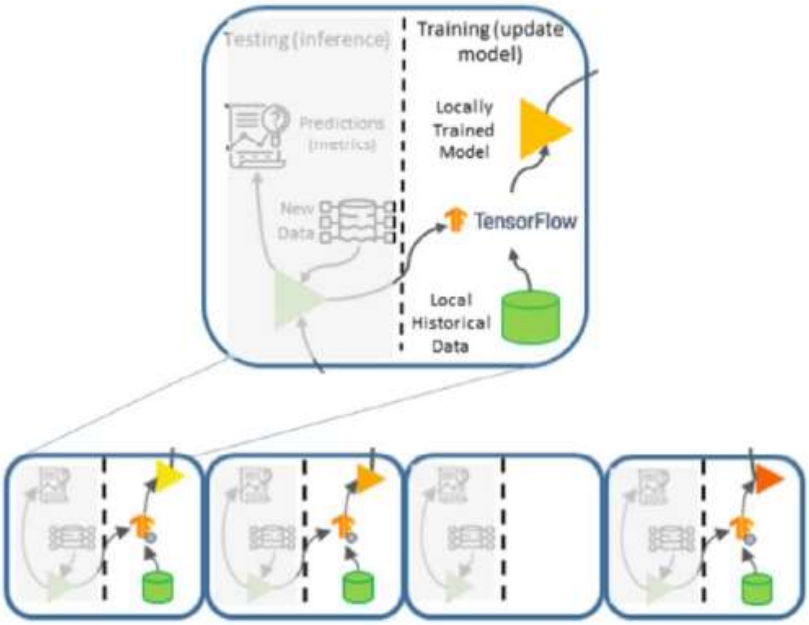



*Figure 5 - Edge node internal workflow (local training)*

### • 2.1.1.2 Centralized Location

The architecture also contains a centralized system (Cloud) that has the following phases:

- Aggregation: The aggregation phase is responsible for updating the shared model. Once each Edge node trains a model using different data (decentralized), the purpose of this step is to aggregate all the generated models. The aggregation method I based upon concepts of Federated Learning. It should be noted that not all ML models must be used for the final aggregated model. There can be filters based on predictive and performance metrics (e.g. standard error) so that only the best models are used.

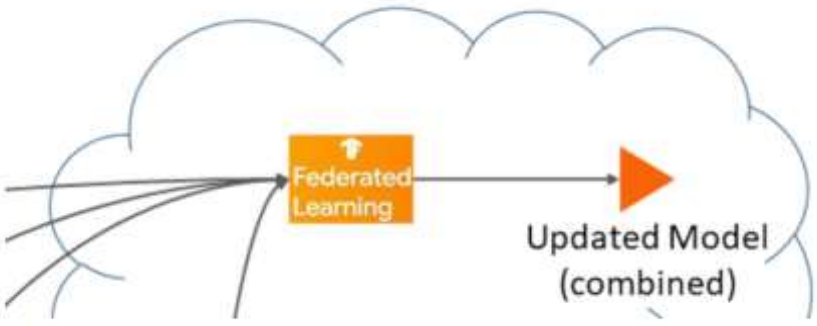



*Figure 6 - Model aggregation on the centralized location*

- Distribution: This phase occurs when there is a new updated model that needs to be shared with all Edge nodes. It consists of sending copies of the new model to each Edge node,

replacing the previous model. The new model will then be used for new predictions in each Edge node.

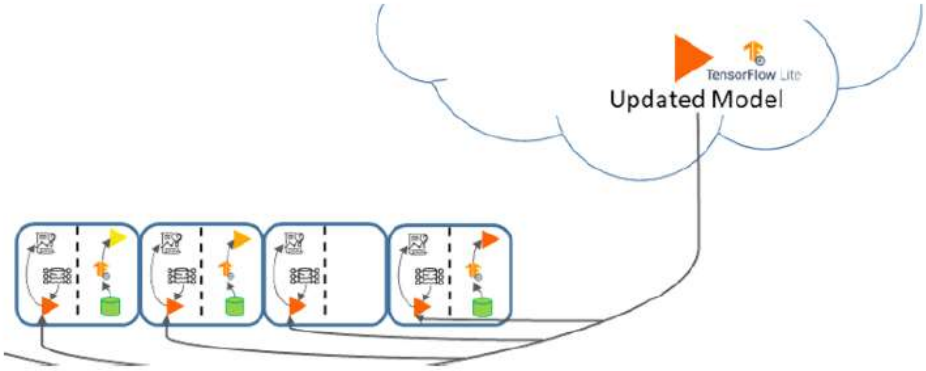



*Figure 7 - Model distribution*

## 2.1.2 Implementation of the Machine Learning Architecture

The Machine Learning architecture presented on section 2.1.1 was implemented on a MEC based on Intel® Smart Edge Open (a framework for managing the 5G components - www.openness.org) clusters, instantiated with Kubedge. There are several Edge clusters and one Cloud (centralized) cluster and communication between them is ensured through a message broker.

The aggregation application is located on the Cloud and uses the broker to send initial models and the aggregates for the prediction applications on the Edges.

The training applications, located on the Edges, send the new models originated from training to be aggregated on the Cloud.

### 2.1.2.1 Technological Architecture of the Solution

Figure 8 depicts the technological architecture implemented:

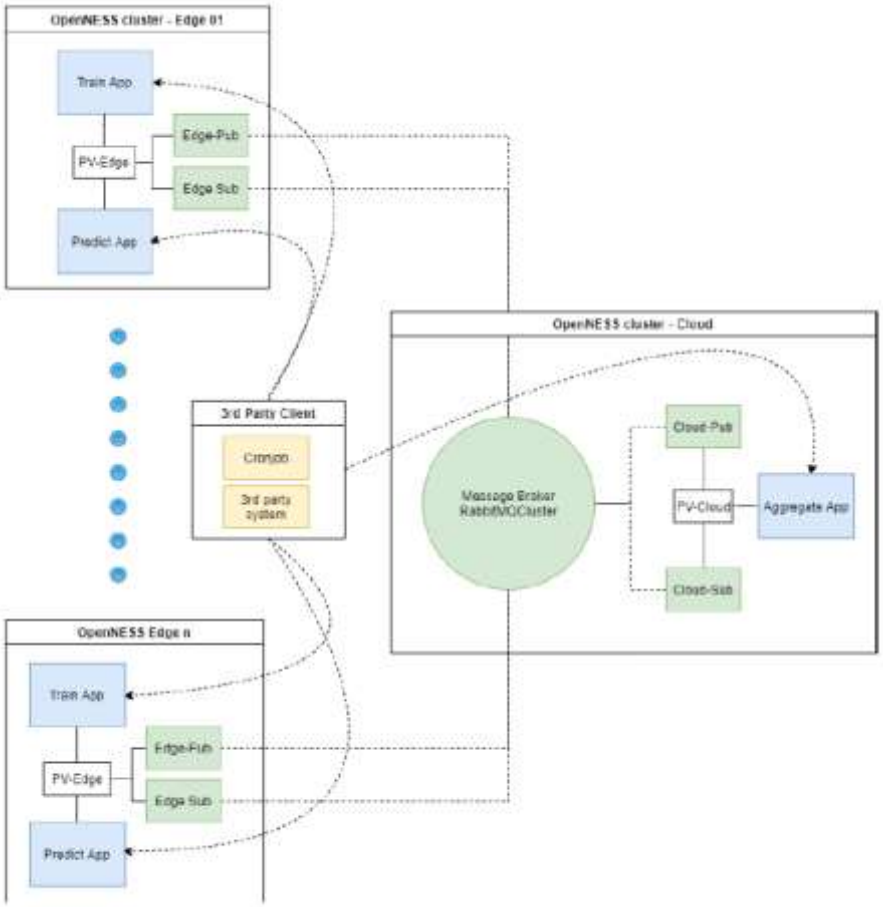



*Figure 8 - Technological Architecture*

On the Edges there are two types of applications:

- Training application, responsible for ML models trained with labeled records;
- Prediction application, responsible for labeling a new set of non-classified records as fraud or not fraud.

On the Cloud there is a ML application that performs aggregation on the Edge models and creates new prediction models.

The solution uses the paradigm Publisher and Subscribe to propagate the ML models through various applications. Both at the Edge and at the Cloud there are applications responsible for publication and subscription of the models. For ease of implementation, it was decided to use a centralized Message Broker service on the Cloud cluster.

The workflow to be implemented is the following:

- First, the Cloud generates an initial model, trained with an existing labeled dataset;
- This initial model is propagated to all the Edges used for prediction;
- Each Edge trains new models as it receives new records and classifications that are then regularly sent to the Cloud for aggregation. These models are not yet used for prediction;
- Regularly the federated application aggregated the new models received from the Edges and generates new prediction models that ate propagated to the Edges;

- The Edges use the models obtained from the federated application to make predictions on whether a call is fraud or not (replacing the previous model). All Edges use a similar model to make predictions that ideally aggregates all the knowledge acquired in all the Edges.

### 2.1.2.2 Publish and Subscribe Paradigm

The Publish and Subscribe is a message communication standard that allows that heterogeneous and distributed services communicate asynchronously. This model allows services to publish and react to events propagated by a Message Broker.

On the Publish and Subscribe there are three types of services:

- Event managers, composed by Message Broker, responsible for filtering, processing and rotating events;
- Event producers, that publish events on the managers;
- Event consumers, that subscribe event types and receive them from managers;

There are several ways to classify events, the most common being the term topic. An event is published on a topic and propagated to all that topic's subscribers.

The Publish and Subscribe has some characteristics of special interest on the distributed architecture of the proposed solution:

- Loose Coupling – the various services of the system (training and federation applications) do not know each other. All communication is performed by the Message Broker, allowing the addition and removal of Edges to the system with minimum configuration costs;
- Asynchronous eventing – Notifications are propagated asynchronously. The events are published at the moment they occur, but their digestion and processing does not have to occur immediately without blocks. For example, when an Edge publishes a new model, it does not have to wait for an answer from the Cloud. Its only concern is if the event was published;
- Fault Tolerance – The two previous points assure the system is resilient and allow, as an example, that parts of the system fail temporarily, be they communication channels or services, without affecting the service as a whole.

## 2.2 ML Framework

TensorFlow Federated is a module focused on Machine Learning models trained on decentralized data. The module was developed with a focus on Federated Learning scenarios, where a centralized ML model is shared amongst multiple clients. TensorFlow Federated currently has two main APIs: Federated Learning, which allows users to apply training and federated predictions with TensorFlow's base models; and Federated Core, focused on users with more experience and that allows the instantiation of new algorithms or implementations of Federated Learning approaches.
One of the classic applications of Federated Learning is related to the learning of smart keyboards (e.g., Gboard from Google), done on smartphones. The next figure exemplifies the process:

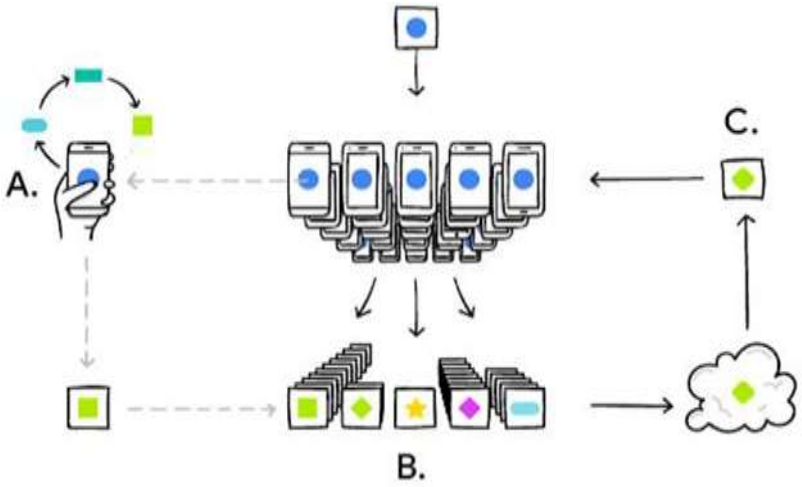



*Figure 9 - Example of a Federated Machine learning application*

Using the example of smart keyboards, each mobile phone trains a model with local data (what its user writes), represented by the letter A. Using learning from several mobile phones, there is an aggregation phase (B). The result is an improved model (C), capable of detecting new words that start to be used by several users or capable of improving suggestions.

The framework needs to be prepared for Federated Learning scenarios similar to Figure X, using various interfaces. One interface is related to Machine Learning models and focuses on serialization and aggregation operations. Serialization allows you to serialize any model into a graph, which is useful when not all devices or edge nodes support running Python environments. Aggregation allows you to join individual models and can be done locally (several models on the same client that use different batches of data) or in a federated way (aggregation of models from different clients). The second interface is called Federated Computation Builders and helps in the evaluation of models. A third and final interface can provide datasets to be used in simulations or experiments with the API.

This was investigated in Activity 3 where will be described in deliverables D3.1 and D3.2.

### ○ 2.2.1 Description

One of the requirements of the Machine Learning architecture is the existence of an initial model that can be propagated to the Edges. Moreover, due to the MEC characteristics of the architecture it was important to have a model with a simple architecture in order to facilitate the deployment phase. In order to answer this criterion a simple 4-layer Neural Network was developed with the use of Keras and TensorFlow.

### ○ 2.2.2 Training Phase

For the Training phase, *binary_crossentropy* was used for loss function, *adam* was used for optimizer and the metric *Area Under the Curve* (AUC). The model can be trained for a maximum of 1000 *epochs*, with an *early stopping* of 3 rounds.

The current version of the Neural Network is depicted in Figure 10Figure 10 along with the corresponding code in.

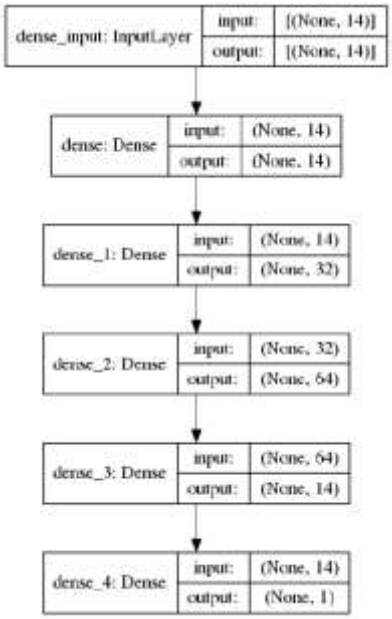



*Figure 10 - Neural Network Architecture*

```
main_model = Sequential()
main_model.add(Dense(14, input_dim=14, activation='relu'))
main_model.add(Dense(32, activation='relu'))
main_model.add(Dense(64, activation='relu'))
main_model.add(Dense(14, activation='relu'))
main_model.add(Dense(1, activation='sigmoid'))
callback = tf.keras.callbacks.EarlyStopping(monitor='loss', patience=3)

main_model.compile(loss='binary_crossentropy', optimizer="adam", metrics=['AUC'])
main_model.summary()
main_model.fit(X_train_main, y_train_main, epochs=1000, callbacks=[callback])
```

*Figure 11. Neural Network code*

As it is possible to observe in Figure 10 and **Error! Reference source not found.**, the developed Neural Network was built with an input layer with 14 neurons (the number of attributes in the dataset after the data preparation stage), 1 output layer with only 1 neuron (corresponding to the binary classification "fraud" or "not fraud") and 3 intermediate layers. The intermediate layers consist of, respectively, 32, 64 and 14 neurons.

### ○ 2.2.3 Federated Aggregation

One of the main concerns of federated learning is the way the various models trained with local (decentralized) data are unified together taking into consideration the condition that there is no sharing of data between clients. There have been many different approaches on this topic proposed in recent years. We selected the Federated Averaging (FedAvg) algorithm for the context of this project (Arcas, 2017).

The FedAvg method was proposed in 2017 and it is still currently one of the most widely used in the context of Federated Learning given its simplicity, ease of implementation and independence from the framework used (even though it is mainly designed for Deep learning algorithms). This method consists in the application of the Stochastic Gradient Descent (SGD) method locally (i.e. in each client) combined with a shared model to generate local models that are then aggregated in a server (typically the Cloud or centralized location identified in the architecture depicted in Figure 1).

FedAvg is built upon the existence of a shared model that is distributed for several clients (Edge nodes in the scope of this project). From the moment the local client has a copy of the shared model, SGD can be used to optimize the initial model with new data from that client. Generating a new local model requires the shared model and the local data as input sources. During the local training each client applies a Gradient Descent step to the existing model using the local data. This results in a new, improved model. Given the low computation costs of SGD each client is allowed to perform this step iteratively during each local training run.

Next, a second phase aggregates the results of all local models generating a new unique model. This stage assumes that a server has access to all local models regardless of the number of models used. To perform the aggregation we use a pondered averaged of all the weights of each local model. The FedAvg expression for the aggregation phase is the following:

$$w_{t+1} \leftarrow \sum \frac{n^k}{n} w_{t+1}^k$$

*Figure 12. FedAvg expression for the aggregation phase*

In this formula, $w_{t+1}$ represents the new aggregated model, $K$ represents each of the clients of the aggregation phase, $n^k$ is the number of records used for training of model $k$, $n$ is the number of total records (sum from all models) and $w^k{}_{t+1}$ represents the local model of client k.

## ● 2.3 Requirements

The Adaptive, Intelligent and Distributed Assurance Platform, AIDA, project aims to deliver an end-to-end 5G-ready fraud management platform that is based on the latest advances in machine learning, edge computing, and hybrid cloud architectures to protect networks for 5G and beyond. AIDA address 5G's scalability and privacy challenges by enhancing RAID's engines and expanding its automation capabilities. In particular, the project will be focusing on the following goals:

- Leverage edge computing and 5G - to distribute RAID platform components to delegate processing to the edge or use central servers, according to the nature of the computation and the type and localization of monitoring and reference data.
- Explore emergent federated machine learning techniques - to learn from local data and push incremental model updates to coordinator nodes that maintain global models based on the contribution of edge nodes and other relevant data sources.
- Test resilience to intrusion or tampering - by requiring the research and application of intrusion detection techniques at multiple levels of the architecture, with the goal of enabling system-wide intrusion tolerance.
- Protect data privacy and confidentiality – by maintaining the confidentiality of the operational data being monitored, analyzed, and protecting the privacy of the entities to whom the data refers.

With this requirements an overview is provided of the 5G Architecture which Raid will interface both at the core and radio access network as well as the infrastructure and technologies that Raid will use to detect 5G fraud.

This section describes a brief review of state-of-the-art Federated Learning frameworks and their limitations regarding use cases potentially involving FL in AIDA.

Federated Learning (FL) is a distributed approach in which the learning of models is pushed towards the network nodes that collect and hold the data. These edge nodes learn directly from the data they collect or hold. Then, they send model updates to a central node (e.g., a server) that orchestrates updates from several participant nodes and shares a global model in the network.

This is an inherently incremental and distributed process, and therefore it is well suited to learn from distributed data and centralized data. The paradigm is to transfer models and model updates instead of the data. This significantly reduces data transfer from the edge to the central systems, lowering latency, reducing storage and network bandwidth requirements, improving the overall quality of service while at the same time providing an additional layer of privacy since data never leaves the origin. The edge nodes receive a shared model from the central node. They learn model updates from the data collected locally and send them back to the central node that orchestrates updates from all the edge nodes.

The highly distributed architecture and network-intensive operation of the RAID platform naturally demands for a distributed paradigm for all types of operation. This obviously includes machine learning algorithms. FL is a natural and even obvious approach to the problem of learning from network traffic data.
Three use cases have been discussed that will eventually involve the need FL:

- Service Disruption Detection

For this use case there is the need for developing a service disruption detection that can work as a micro service and is able to monitor the overall health of both Platform and OTT Service ability to provide the service according with the contracted terms.

- Platform Service Abuse

Development of a service abuse detection at the platform level, that can work as a micro service and is able to monitor the monitor subscriber service connection and usage patterns, together with the service contract data, in order to detect situations of abnormal usage that don't comply with the contracted service terms or have the risk to have the CSP incurring in financial or reputational losses. If such a case is detected, relevant data for the analysis should be collected and an alarm generated. Automated actions may be executed for certain scenarios.

- OTT Service Abuse

Development of an OTT service abuse detection, at the platform level, that can make use of an wealthier set of data not at the reach of the OTT Service provider, and that can work as a micro service in order to monitor the connections to the OTT service and usage patterns, together with the service contract data, and spot situations of abnormal service usage that don't comply with the contracted service terms or have the risk to generate losses to either the CSP or the OTT Service provider. If such a case is detected, relevant data for the analysis should be collected and an alarm generated. Automated actions may be executed for certain scenarios.

From the machine learning perspective there was the need to focus on a specific instantiation of each use case. The following have been adopted
Use case 1: DDoS early detection
Use case 2: Phone fraud detection
Use case 3: TV piracy service detection

Each of these have particular requirements at the algorithmic level, however, for the purpose of framework development, they share common requirements

- Ability to perform distributed learning
- Ability to learn online from complex data (networks, high-speed streams) in real time
- Ability to perform inference with low latency
- Data privacy guarantees

## 2.3.1 Framework Requirements

The adequacy of FL frameworks has been studied under the following perspectives:

- License: the usage and distribution license must be open-source and business-friendly
- Ability to operate in a distributed networked environment: RAID's architecture is complex, based on a large number of heterogeneous nodes orchestrated in a Kubernetes environment. It is important that the framework has high modularity, portability, and a lightweight

footprint given eventual computational limitations of nodes. It is also required that the framework includes network connectivity using standard network protocols.

- Ability to operate in the laboratory: although the final production environment is rather complex, the framework is also required to operate in very simple setups (even single machines), typically used in laboratorial settings for offline algorithm development and validation.
- Package maturity: although there is a very large number of publicly-available FL frameworks, most are very recent and hold potential risks in adoption due to uncertainty of continuity.
- Ease of implementation and extensibility: given that ML researchers are not typically proficient with the inner-workings of large distributed systems, it is important that the framework provides clear and easy-to-use abstractions of this eventual complexity.
- Quality of documentation, for obvious reasons.

From the above analysis, the decision was to adopt Flower[1], an open-source python-based FL framework.

Flower is a framework for building federated learning systems. It can be used with any machine learning framework, for example, PyTorch and TensorFlow. Flower's API is mainly composed by a client and a server. The Flower server is the communication channel between two or more Flower clients that compose the FL environment. One round of the FL process, which is repeated for multiple rounds, goes as follows. The server sends model parameters to the clients. The clients run the training and update the parameters. The updated parameters are sent back to the server which averages all received parameter updates.

It is licensed under the Apache 2.0 license, has been developed and maintained for 2 years and is currently very active (7 releases, over 25 developers, more than 150 forks). It follows a client-server architecture using standard TCP/IP, it is well documented and it is extremely simple to install, extend and operate in simple systems, while still being sufficiently flexible to be deployed at scale on the cloud using state-of-the-art system orchestration environments based on Kubernetes.

#### • 2.3.1.1 Federated learning architecture

Following the above considerations, AIDA's consortium partners have agreed on a high-level FL architecture. This architecture is illustrated in Figure 11.

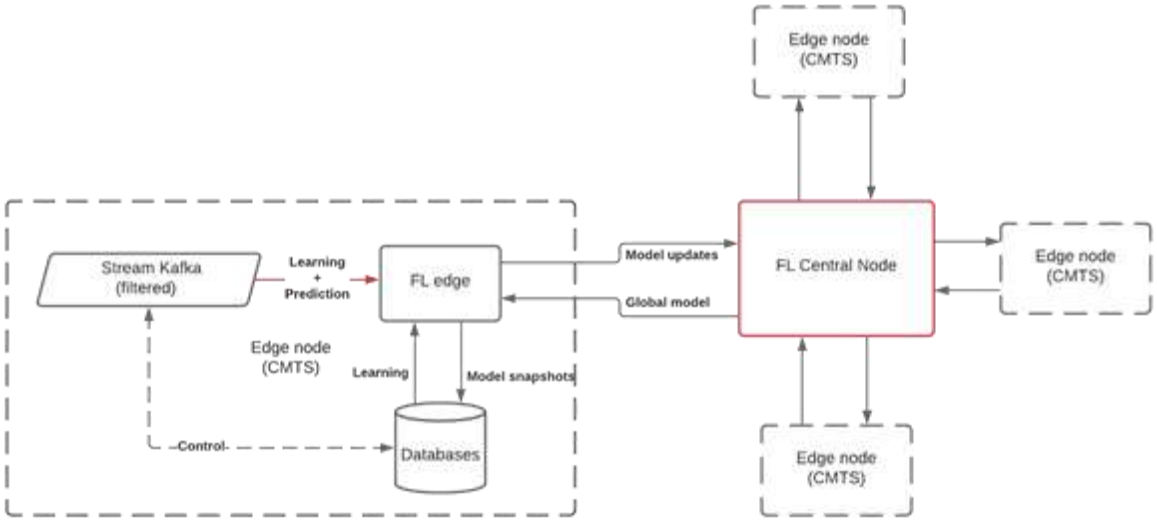



*Figure 11 - High-level Federated Learning architecture*

As expected in a FL framework, the training process involves a central node and multiple edge nodes. Task 3.1 of AIDA focuses on the development of a framework capable of coordinating this process between central and edge nodes. Since several state-of-the-art frameworks already provide the essential mechanisms for this coordination, the biggest challenge in AIDA is concentrated on the inner-workings of edge nodes, in particular efficient access to streaming data, as well as local and remote databases. Regarding streaming data, a natural intermediary between raw traffic data and the learning algorithms at the edge is Kafka, a popular data stream management platform, capable of some online pre-processing. Besides Kafka, it is expected that the framework has access to other data sources available on the network.

## 2.4 Platform communication

By design, each micro service needs to be deployed individually, describing the interfaces and dependencies, conforming to the Representational State Transfer (REST) software architectural style. A RESTful web service, exposing a RESTful API that provides its peers with a constant endpoint for the service being provided.
The application that are executed on the edges need to have at least these five API's groups implemented:

- Case API
- Machine Learning Update API
- Machine Learning Test API
- Edge Monitoring API
- Edge Performance API

**Case API Description:** Cases are opened in the cloud by each Edge Location, implying the transmission of details which will allow the analyst to review and classify it. Key Areas: -Data at rest; Data in transfer; Anonymization of sensible data.

**Machine Learning Update Description:** Each edge location will have its own model trained with local data, this federated model will be shared with the cloud whenever it is updated. The cloud will produce a new model with the combined.

**Machine Learning Test Description:** Monitor the performance of the ML model and benchmark with previous versions identifying performance issues.

**Edge Monitoring Description:** Monitor the health of the Edge Location resources insurance that it's live and performing under expected the thresholds, allowing auto or remote configurations to be deployed, including reference data.

**Edge Performance Description:** Each Edge Location needs to be monitored in terms of business performance, e.g.: there are no data feeds to be processed, or they have wrong formats, how many cases per minutes are opened, closed, high variations on trends or silence systems even though no errors being reported.

For further analysis, this was investigated in Activity 1 and described in deliverable D1.2 where was described the platform architecture and APIs required to report a new fraudster, update a fraud model or for monitoring and performance.

## 2.5 Security

The strategy proposed operates at multiple levels of the system. For privacy there are mechanisms that intend to assure that information is processed and stored in a manner that reduces probability of private information leaks and minimizes the risk of data re-identification. The mechanisms proposed address privacy challenges through trusted execution environments that allow the information to be processed by machine learning algorithms in a privacy-preserving distributed manner. Another important aspect is the development of a framework that allows to test different configurations and assess mechanisms according to privacy and utility levels of data. These mechanisms operate at the cloud and edge levels.

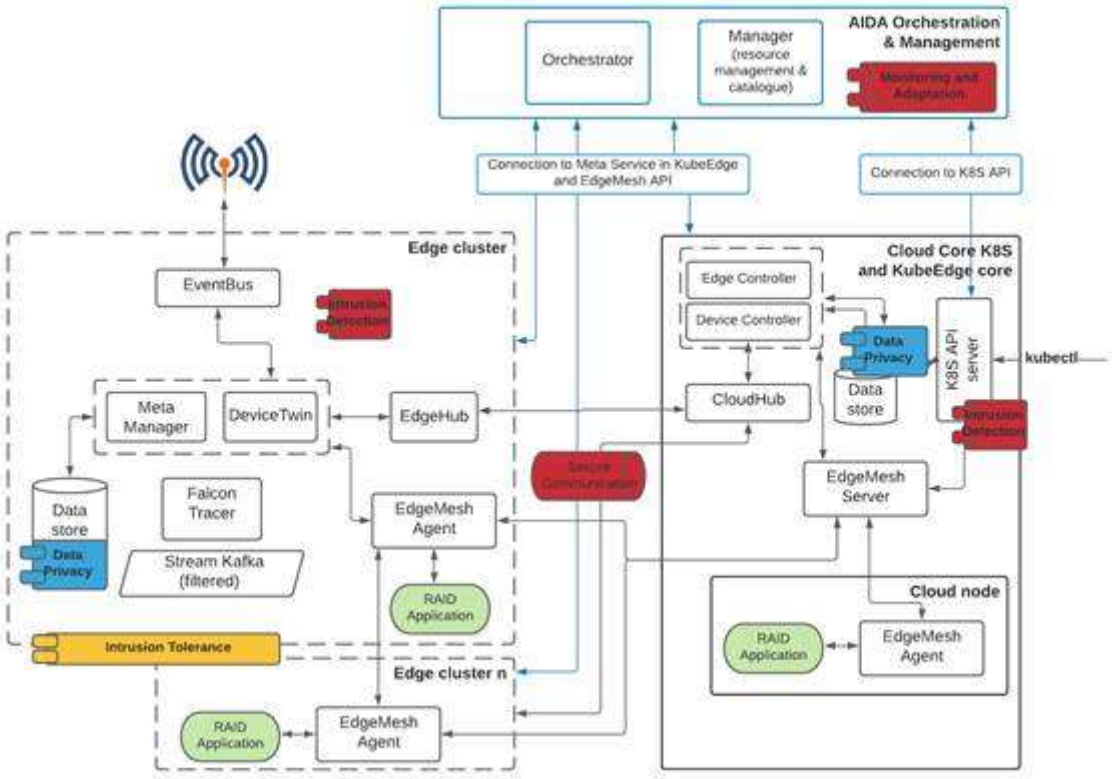



*Figure 12 - AIDA security and privacy design overview.*

For security purposes there are three main mechanisms applied to the AIDA platform:

At the cloud and the edge there are mechanisms for intrusion detection and tolerance that aim to prepare the system to detect security breaches and direct security countermeasures to reduce and mitigate the possible damage caused to the affected components.

The intrusion detection mechanisms are directed to microservice architectures that are used at the cloud and the edge, collecting host information on each service and building profiles to detect deviations from the normal execution of the system. For intrusion tolerance purposes, the system will become capable of maintaining operation regardless of security breaches.

To assure that cloud and edge levels can communicate securely, the security strategy proposes the use of encryption mechanisms that assure integrity, confidentiality, and privacy between the diverse components at the edge and cloud. Further, AIDA will also be supporting scalable authentication and authorization mechanisms, relying on OpenID Connect solutions.

To integrate all the components of the security strategy, AIDA will have available a monitoring and adaptation module. This module will be responsible for collecting data about the platform and perform modifications in its configurations or mechanisms used to accomplish the goals of the platform. This module will be deployed together with the AIDA orchestration and management component that orchestrates the different levels of the platform.

**a) Secure Communications**

Integrity, confidentiality, and privacy in AIDA is supported using Transport Layer Security (TLS), to encrypt all the communication between the components of the AIDA architecture. The services,

exposing functionalities through HTTP based APIs can encrypt the communications relying on HTTPS, which heavily relies on TLS.
The secure communications can rely on Service Mesh architectures, which manage the required elements (certificates) to support mutual TLS in a transparent fashion. The service mesh architecture relies on proxies (often called 'sidecar proxies'), which are deployed in front of micro services and are responsible to provide infrastructure services (e.g., load balancing, ingress controller, egress controller) for applications based on micro services.

**b) Authentication and Authorization**

OpenID Connect is used as the solution for authentication and authorization in AIDA, with the advantage of supporting federated scenarios as illustrated in Figure 13. The envisioned scenario considers a user with multiple devices hosting diverse applications, which are identified as User Agents. The user needs to have the means to express its levels of trustworthiness in the diverse devices and respective applications. As a matter of example, a user may tend to trust more on the iPhone device given the full control of Apple in terms of hardware and software, in comparison to a Tablet running Android. In addition, Google Chrome and Safari user agents can be installed in the iPhone, with the tendency for the user to trust more on the Safari user agent. This leads to the need of identifying devices and applications in a unique and compatible fashion with Federated Identity Management (FIM) solutions.

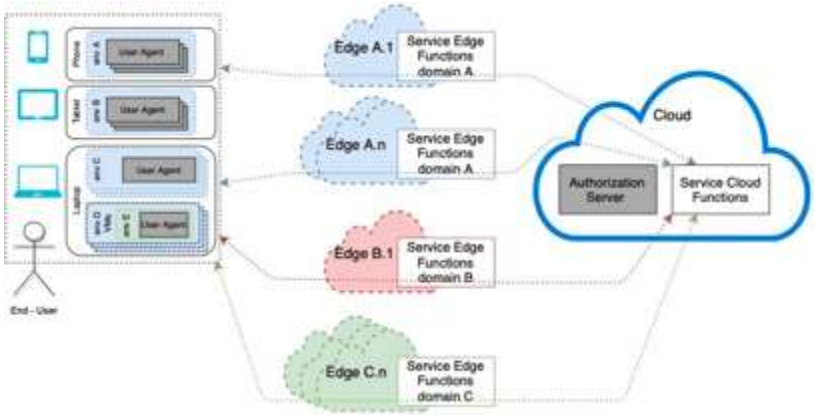



*Figure 13 - Example of use case for authentication and authorization.*

OpenID Connect (OIDC) is a feasible FIM solution, relying on the OAuth 2.0 protocol that is widely used for dealing with the authorization process to access resources (e.g., application APIs requiring access to information fields of the user, like email address).

The OIDC core architecture considers, mainly three entities: the authorisation server, also known as OpenID Provider (OP) and is responsible to authenticate an end user and to obtain the end user consent/authorization; the client, also known as Relying Part (RP), which is responsible to make the interface with the end user to manage the required information for authentication; the user agent, commonly in a browser, interacts with the client to provide the required information, like the user consent.
OIDC introduces the concept of scopes to request access to claims. For instance, the RP may ask for some user attributes (e.g., scope=profile name family_name), which the user can consent to. Each

scope returns a collection of these attributes, referred to as claims (scope=email address phone). To identify an OpenID Connect flow, the scope of openid is required (e.g., scope=openid).

OIDC adds the ID Token extension to OAuth 2.0, which is a security token with several information fields - claims to authenticate an end user. Several claims are specified in OpenID Connect, such as the issuer identifier (iss) which is a kind of URL with the scheme, host and port number, the subject identifier that is unique and with local context for the end user (sub), the audience (aud) which defines to whom the ID token is suited to, it corresponds to the OAuth client_id.
The trust relation of the user with a certain service, does not consider the environment, device where the access to a specific service is performed, as described earlier. To overcome this issue AIDA introduces an extension to OpenID Connect – AIDA Context Information (ACI), in a standard fashion to assure interoperability with existing OpenID Connect implementations.

### c) Edge and Cloud Security

The edge and cloud continuum is assured through the interaction between kubernetes platform and the kubedge, as documented in D2.1. Both Kubernetes and Kubeedge have been identified as being feasible solutions for edge-cloud continuum. The security is enabled in two different perspectives: control plane, and data plane, as documented in the following subsections.

The edge-cloud continuum is assured by the EdgeCore (EdgeHub subcomponent) and CloudCore (CloudHub sub-component), respectively. The control plane of these components is secured - edge-cloud security continuum through the following mechanisms:

- HTTPS configured with TLS.
- Token to employed for authentication between the cloud and edge

Data plane security mainly relies on the communication between microservices that run at the edge and cloud sides (EdgeMesh, 2022). EdgeMesh has the advantage of providing the benefits of service mesh, and can be configured at the Kubernetes side through the helm package system for kubernetes.

### d) Intrusion Detection

Three approaches are being evaluated to enable detection of intrusions in scalable and elastic microservice-based systems. These approaches intend to enable intrusion detectors to deal with the presence of multiple service replicas, a common aspect of microservice-based systems. Current methodologies do not address security threats looming over a set of multiple replicas of a service. Instead, they tackle the issue focusing on each unit, thus requiring a higher number of active profiles.

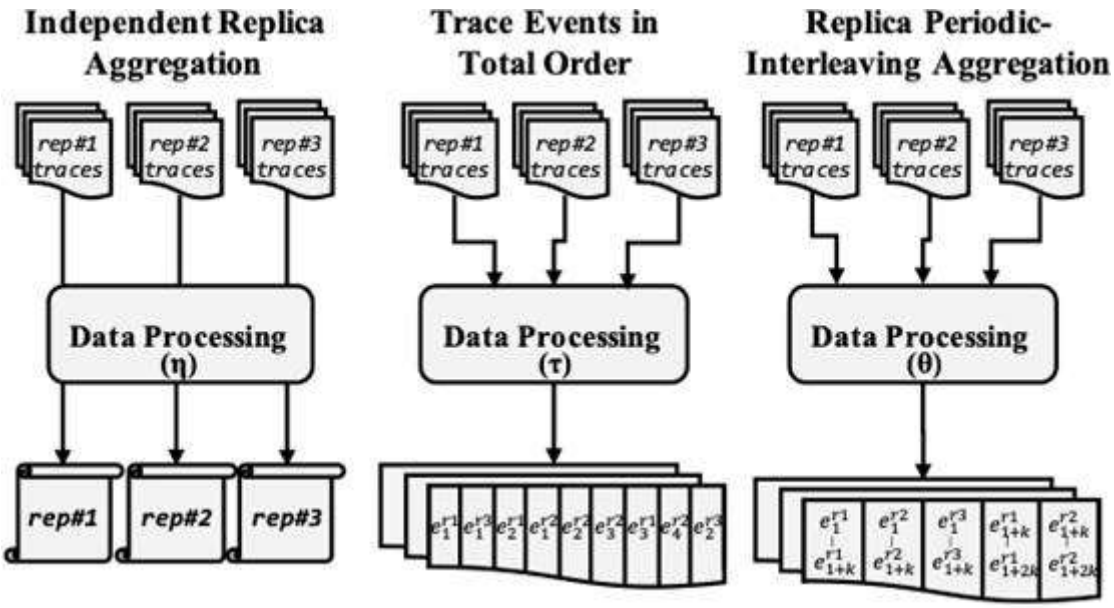



*Figure 14 - Overview of the proposed approaches for data processing for micro services intrusion detection.*

The approaches integrate with a typical anomaly-based intrusion detection methodology, as presented in Figure 15 that works on two main phases: the **training phase,** where the operational profiles are learned, and the **detection phase,** where events are analysed for intrusions. During the training phase, classifiers use benign information from the system to create its benign behavior profiles. These profiles are used during the detection phase to identify deviations from the normal behavior learned and raise alarms.

The data processing approaches operate over the data collected and feed it to intrusion detection algorithms. Consequently, regardless of the number of active replicas, the profiles can continuously be used to detect security intrusions with effectiveness and assure uninterrupted security of service replicas. This ability removes the necessity for multiple training conditions to cover all the deployment scenarios faced during detection phase; thus, promoting reusability of profiles and generalization of the knowledge acquired. Variations in load and demand, and consequently auto-scaling operations performed, do not impair the capacity of profiles to detect malicious events.

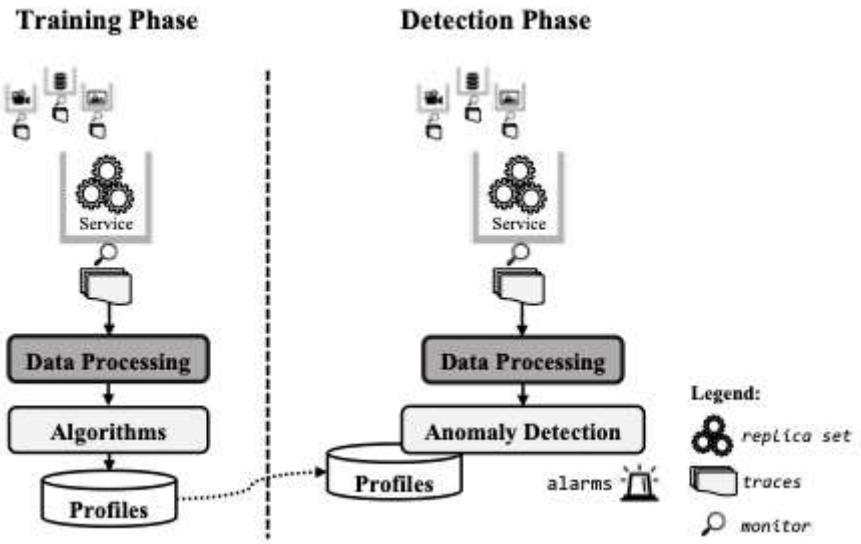



*Figure 15 - Integration of our approaches with an anomaly-based intrusion detection methodology for microservice-based systems.*

**e) Privacy Mechanisms**

SOTERIA is a system for distributed privacy-preserving machine learning, which leverages Apache Spark's design and its MLlib APIs.

Our solution was designed to avoid changing the architecture and processing flow of Apache Spark, keeping its scalability and fault tolerance properties. As depicted in Figure 16 by the gray boxes, a Spark cluster is composed of a Master and several Worker nodes. Before submitting ML tasks (e.g., machine learning training and inference operations) to the Spark cluster, the user must load its local datasets and models to a distributed storage backend supported by Apache Spark.

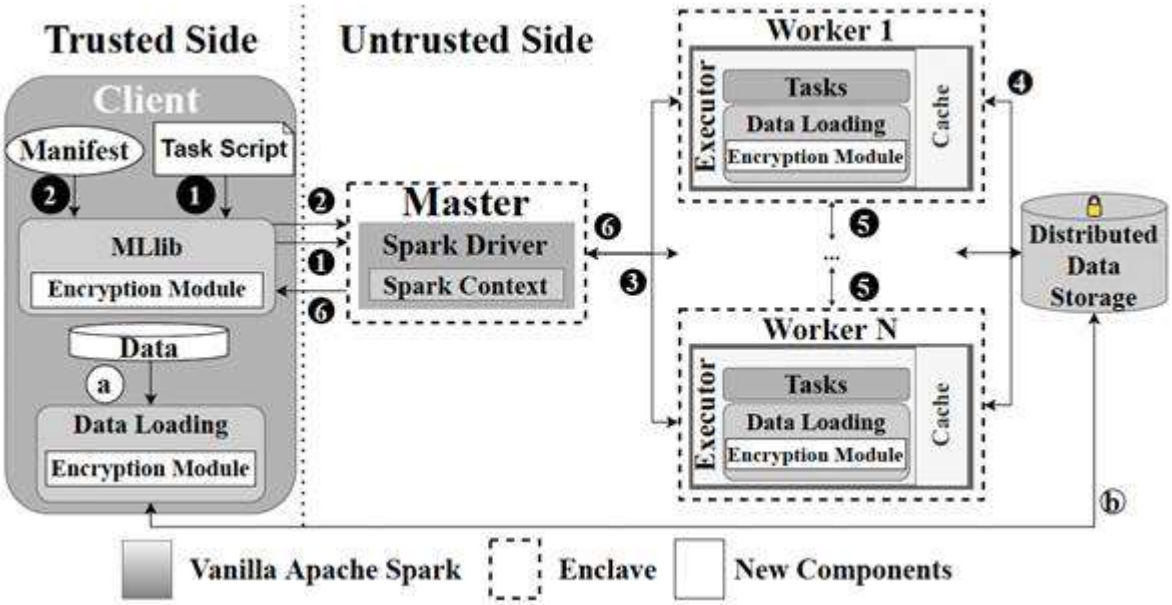


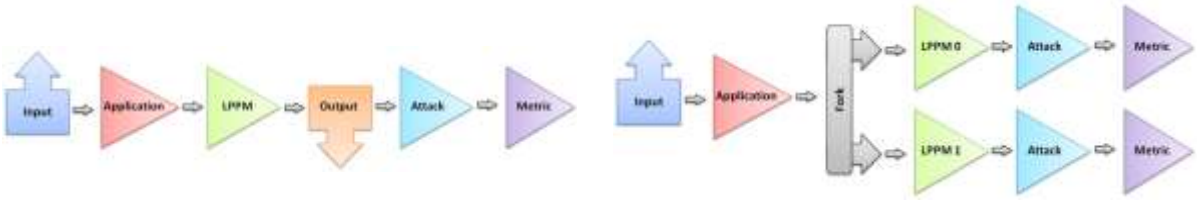

*Figure 16 - SOTERIA architecture and operations flow. Main components of Apache Spark vanilla are depicted in gray boxes, whilst dashed boxes represent the components inside enclaves and white boxes depict the new components implemented in SOTERIA.*

After the data loading step, the user can then submit ML processing tasks to Spark's client that is responsible for forwarding these tasks (scripts) to the Master node. Namely, tasks are submitted to the Spark Driver component which generates a Spark Context allowing access to the resource manager and then distributing the tasks to a set of Worker nodes according to its needs. Therefore, the Spark Driver must have direct access to the computations, processing logic, or ML task scripts, before delegating the tasks to the Workers to optimize resources' usage.

**f) Data Privacy**

The lack of a standardized manner of quantifying privacy as well as the limitations of the existing frameworks conduct us to develop a novel privacy framework to enable the implementation and evaluation of diverse anonymization methods suitable for the project requirements.

The main objective is to develop an extendable privacy framework that allows to test configurations, apply Privacy-Preserving Mechanisms (PPMs) and assess mechanisms according to the achieved privacy and utility level of data. To do so, the privacy framework will be proposed and available as an open-source Python package, similarly to other well-known scientific toolkits (e.g. scikit-learn and scikit-mobility). Based on the state-of-the-art analysis, quantifying privacy commonly follows a pipeline composed by the steps: input data, PPM, attack, metric, and output data (c.f.Figure 17). Since the schedules presented in Figure 17 are merely representative, Figure 18 presents a real world experimental methodology, following the same pipeline concept (i.e. input data, PPM, attack, and output data). Concerning the input data, beyond reading, data might need to be processed, as observed in the followed methodology. From Figure 18 , we can also highlight the fact that three PPMs were applied, which implies testing and configuring three different mechanisms. Systematizing this process constitutes one of the motivations of proposing a privacy framework.




(a) A simple schedule (with intermediary output after LPPM)

(b) A simple branching schedule

*Figure 17 - Examples of schedules from Quick Start Guide (Shokri et al., 2012).*

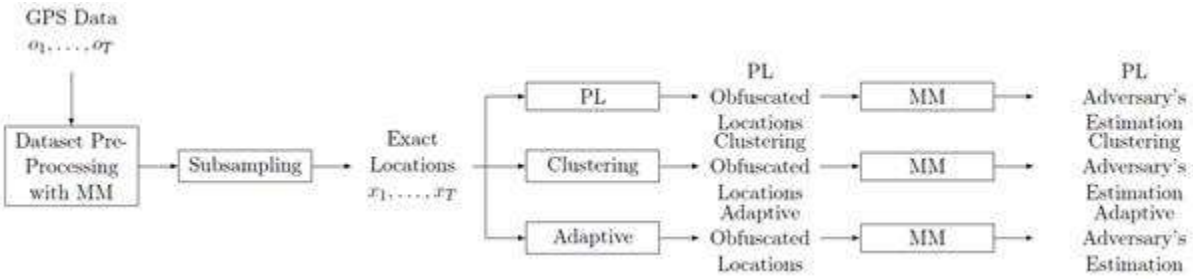



*Figure 18 - Scheme of a methodology from (Cunha et al., 2019).*

Figure 19 presents a schematic of the privacy framework components, where each subpackage represents a step of the pipeline. Each subpackage contains the corresponding adapter, which is an abstract class that can be extended by implementing the abstract methods (i.e. relevant methods for the component). These adapters make the framework easily extendable, by allowing the implementation of new features (e.g. new PPMs or metrics). The subpackages presented in Figure 19 can be briefly described as follows. The data subpackage is responsible for handling a data type by providing methods for specific tasks such as the ones represented a grey in the figure: read data, process data, filter data, and visualize data. For adding new data types, it is just needed to create a new class that extends from the Data Adapter and implement the respective functions, being possible to add new functions to handle the new data type. The remaining subpackages are designed in the same way. Thus, for adding a new PPM, attack or metric, we just need to extend the correspondent adapter and implement it according to the desired methods/requirements.

Considering location data as a use case, the designed privacy framework will include implementations of appropriate PPMs, attacks, and metrics in this context. The features already implemented are represented in the scheme of Figure 19. For instance, Geo-indistinguishability (Geo-ind), Adaptive Geo-ind, and Clustering Geo-ind constitute the PPMs that are already implemented and available in the privacy framework. By providing these implementations, we are not only systematizing the concept of quantifying privacy, but also promoting it from a practical point of view and motivating for an important scientific principle: reproducibility.

As aforementioned, one of the major concerns when designing the privacy framework is its extendibility, such that it can support heterogeneous data types, as well as different types of PPMs, attacks, and metrics. Taking the goals of the AIDA project in mind, we will be able to consider PPMs suitable for the needs of the project, by implementing PPMs, metrics and attacks thereof. After making the Python package publicly available, we intend to make the privacy framework accessible through a Progressive Web App (PWA), that is, a web application that can be used both online or offline to perform experiments and analysis over data.

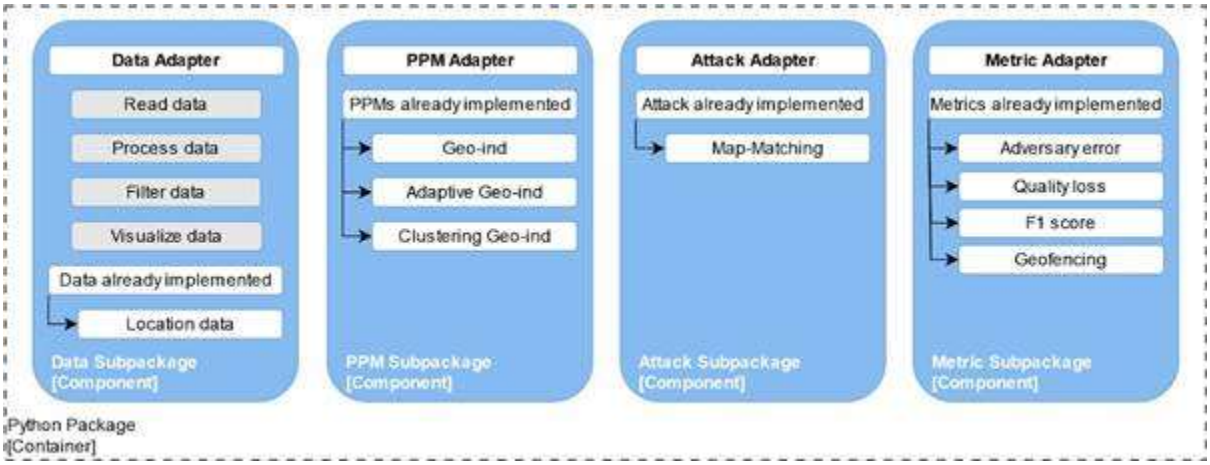



*Figure 19 - Scheme of the privacy framework components.*

**g) Monitoring and Actuation**

As part of the self-adaptation process of microservice-based systems, two important activities are needed: 1) monitor the systems and 2) act on the systems to achieve a predefined quality standard. These are the two interaction points of the adaptation control loop (IBM, 2005) needed to promote self-adaptation abilities in the managed system. Two common approaches to promote self-adaptation in a managed system: through an external layer of control or by adding self-adaptation mechanisms directly to the managed system itself (Salehie, 2009). The approach that will be used in AIDA promotes self-adaptation through an external management layer to separate the concerns of the business logic and self-adaptation abilities from the steady-state functioning of the system, which in turn support reasoning about the quality and generality of adaptation mechanisms and also reuse across systems (Garlan, 2004).

In the context of AIDA Orchestration and Management, the TMA framework (http://tma.dei.uc.pt) will be used. TMA is the Trustworthiness Monitoring and Adaptation framework, which has a microservice architecture and can be deployed in a Kubernetes cluster. It uses an external component to promote self-adaptation in the managed systems (Pereira, 2020). Figure 20 shows its architecture.

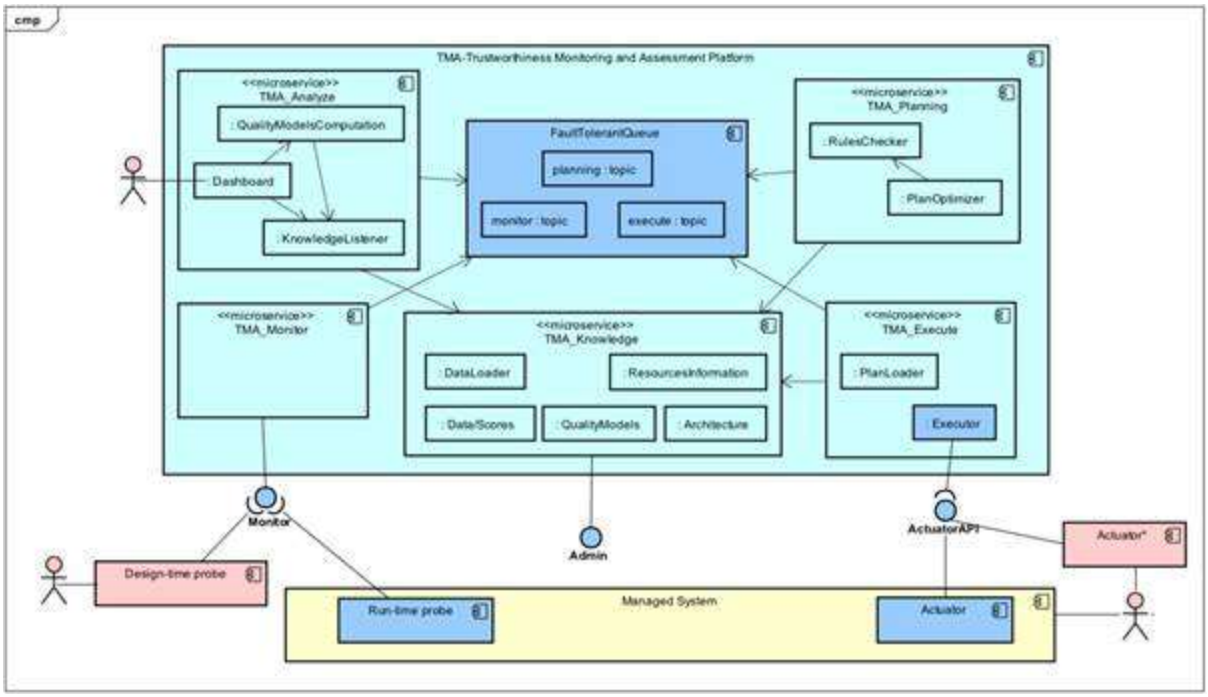



*Figure 20 - TMA architecture and the managed system being monitored and adapted by TMA (adapted from Metrics Project).*

TMA is based on the MAPE-K adaptation control loop created by IBM (IBM, 2005). MAPE-K stands for Monitor, Analyze, Plan, Execute, and Knowledge and they represent activities of the feedback control loop. TMA provides a component for each one of these activities. Monitor provides a REST interface to receive the data of the managed system. All the data is received by the Monitor component, and all the information needed by TMA is stored in a database through the Knowledge component. Analyze is responsible for reasoning over the data and creating metrics/indicators from the collected measurements. The Plan component comes up with an adaptation plan if an adaptation is needed (e.g., the CPU usage exceeds a threshold). Finally, the Execute component receives the plan and executes it by invoking operations to promote changes in the managed system.

For further analysis, this was investigated in Activity 4 where was described in deliverables D4.1 and D4.2.

## • 2.6 External validations

External validations are necessary so we have additional inputs to the framework requirements and execution. A 5G lab was identified partnering with a telecom operator, this will give the pilot a testing environment that will be similar to a real telecom network.

# 3. Integration plan

## • Project Activities - Gantt

**Activity 1: Requirements and Design analysis**

- T1.1 - Definition and Analysis of requirements for the Pilot execution
- T1.2 - Solution design validation

**Activity 2: Solution components implementation**

- T2.1 - Implement Monitoring on edge architecture
- T2.2 - Implement Federated Machine learning framework on edge architecture
- T2.3 - Create solution deployment scripts
- T2.4 - Develop identified Use cases Machine Models

**Activity 3: Solution Deployment**

- T3.1 - Deploy components architecture
- T3.2 - Security controls/components deployment
- T3.3 - Solution dry run deployment
- T3.4 - Local and Cloud deployment of the pilot

**Activity 4: Test and Validate solution**

- T4.1 - Specification and test execution to identify defects
- T4.2 - Solution components test and validation
- T4.3 - Evaluate prediction performance on real data

*Figure 21 - Project Activities Gantt*

# 4. Validation and Quality Assurance

Quality assurance (QA) testing is the process of ensuring that your product or service is of the highest possible quality for your customers. QA is simply the techniques used to prevent issues with your product or service and to ensure great user experience for your customers.
In AIDA, the main features will use private and decentralized data, so quality assurance as an important role to guarantee that all the security mechanisms are correctly implemented.

Tests execution should be done in an appropriate environment gathering the analysis of the results obtained.
In order to have the best tests results and have the best coverage of all the developed components is to test using a real fraud use case. Implement a way to detect a fraud in telecommunications known as International Revenue Share Fraud (IRSF), using Edge Computing and Machine Learning. One of the main objectives is to understand whether a certain telephone call is included in this type of fraud or not, and we intend to develop a solution capable of doing so with the following features:

- In the telecommunications base station, the Edges of the solution, telephone calls occur between two (sometimes three, for example, in the case of forwarded calls) telephone numbers. Each of these calls gives rise to a record that contains a set of information related to the call, being of special interest to the solution the originating number, the destination number, the duration of the call and its cost. For privacy reasons, these records cannot be transmitted to systems external to the base station;
- A classification (target column) is added to the telephone call records present at each Edge by an external labeling process;
- It is intended that in each base station there is a system based on Machine Learning capable of learning to classify telephone calls as fraudulent or not, based on the classification obtained by the labeling process;
- After learning, it is intended that in each base station, there is a system based on Machine Learning, capable of detecting if a telephone call is fraudulent or not, namely if it is an IRSF fraud, using the trained models;
- It is intended that there is a transmission of knowledge between Edges, of what was learned by each Edge Machine Learning model on IRSF.

Based on these assumptions, an IRSF fraud detection system was designed based on Machine Learning and federated learning. Federated learning unifies several Machine Learning models, without needing to know the records that trained each of the individual models, thus allowing to respect the privacy requirement that exists in this project.

The main focus will be to guarantee that the components are used, the results provided are correct and have a secure communication.
A full strategy of how security and privacy will be enforced in the platform, communications and data is detailed in deliverable D4.1 and D4.2. It's advisable to read chapter 2 related with secure communication and Edge-cloud security described in deliverable D4.2.

One of the most underrated aspects of creating your Machine Learning Model is thorough validation. Using proper validation techniques helps you understand your model, but most importantly, estimate an unbiased generalization performance.
The content fraud use case will be validated using ground truth, checking the results of machine learning for accuracy against the real world, since this is implicit in datasources.

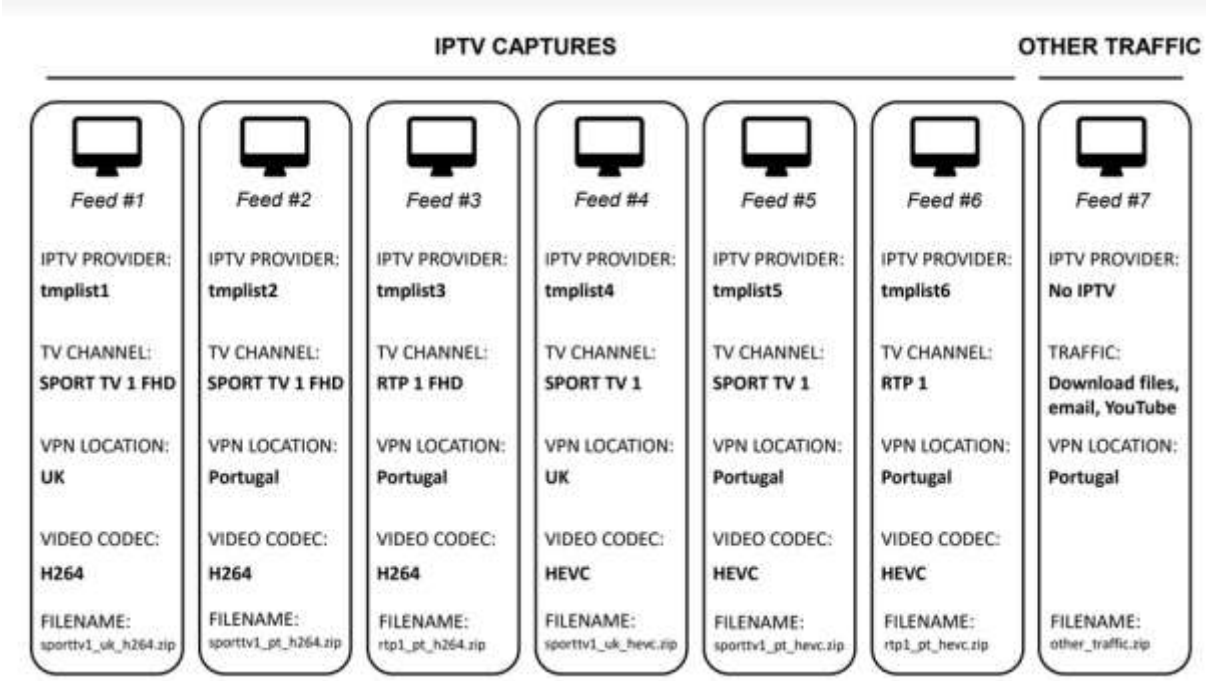



*Figure 22 - Datasource feeds for machine learning models validation*

Successful implementation of validations will identify Feed #1 is equal to Feed #2, Feed #4 is equal to Feed #5 with a high assertive level (>=90%) and with enough interval distance from the non-related Feeds. 5 minutes of traffic should be enough to validate the feeds content and return an evaluation.

When compared with existent algorithms for telecom fraud detection Gen2Out outperforms them in detection assertivity, speed and scalability. This will be supported by datasets with classified data.

It's advisable to read deliverable D3.1 related with machine learning detection algorithms for fraud uses case.
A framework is a set of components working together. The main intention behind a framework is to facilitate (app or service) creation. Framework performance/benchmarking evaluation helps identify gaps in process, strategies, and techniques to achieve your goals.
For further analysis, this was investigated in Activity 3 where was described in deliverable D3.1.

# 5. Conclusions

This document presents the security strategy along with the security and privacy requirements for the AIDA project. An extensive review of the state of the art is performed focusing on the main concerns of each task of the project and the relevant topics addressed in each one of them. This

review is based on the scope and security concerns of the AIDA platform taking into account the architecture and technologies adopted. The main threats are identified along with the stakeholders and main actors, so as to pinpoint the major attack venues.

With the requirements identified the platform will comply and have assurance of its ability to operate securely and maintain the privacy of the customers and their data during the operation. Focusing on topics such as the secure communication between AIDA components that can be distributed throughout the edge or the cloud, allows a secure exchange of data maintaining its integrity and privacy. Further, the pilot also covers the detection of fraud using a well know type of fraud With regard to data privacy, it is expected to provide a privacy-preserving framework for machine learning workloads as the information can be processed in potentially untrusted environments and also contribute with a framework for preserving the privacy of heterogeneous data types, given the nature of the environments analyzed by the AIDA platform.