Research and Innovation Action
HORIZON-CL6-2022-GOVERNANCE-01
TOPIC: HORIZON-CL6-2022-GOVERNANCE-01-11

AgriDataValue

Smart Farm and Agri-environmental Big Data Value

Deliverable D2.1
AgriDataSpace Underlying Technology

Authors	I. Oikonomidis, I. Chrysakis
Nature	Report
Dissemination	Public
Version	v1.0
Status	Final
Delivery Date (DoA)	M12
Actual Delivery Date	31/01/2024

Keywords	AgriDataValue, Data, DataSpaces, Integration, Verification, Validation, Deployment, FDML, Storage, XAI, Secure
Abstract	This handbook describes AgriDataValue Platform first version. It provides an overview of the deployment options that AgriDataValue has chosen, explaining how these options favour the platform's reproducibility as well as enforce its resilience and performance capacity. Finally, it reports the platform and its component verification and validation plan. In addition, the document describes in detail the design and functionality aspects of the AgriDataValue Platform components. It also discusses how the various components get integrated into a coherent framework offering coordinated services.

Funded by the European Union under Grant Agreement no. 101086461

ACKNOWLEDGEMENT

The AgriDataValue project is funded by the European Union under Grant Agreement No. 101086461. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Executive Agency, while neither the European Union nor the granting authority can be held responsible for any use of this content. No part of this document may be used, reproduced and/or disclosed in any form or by any means without the prior written permission of the AgriDataValue consortium.

	Participant organisation name	Short	Country
01	SYNELIXIS SOLUTIONS S.A.	SYN	EL
02	ATOS IT SOLUTIONS AND SERVICES IBERIA SL	ATOS	ES
03	SIXENSE ENGINEERING	SIXEN	FR
04	NETCOMPANY-INTRASOFT SA	INTRA	LU
05	SIEMENS SRL	SIEM	RO
06	SINERGISE LABORATORIJ ZA GEOGRAFSKEINFORMACIJSKE SISTEME DOO	SINER	SI
07	ALMAVIVA - THE ITALIAN INNOVATION COMPANY SPA	ALMA	IT
08	INTERNATIONAL DATA SPACES EV	IDSA	DE
09	SOFTWARE IMAGINATION & VISION SRL	SIMAVI	RO
10	SINGULARLOGIC S.A.	SLG	EL
11	EIGEN VERMOGEN VAN HET INSTITUUT VOOR LANDBOUW- EN VISSERIJONDERZOEK	EV ILVO	BE
12	ETHNIKO KAI KAPODISTRIAKO PANEPISTIMIO ATHINON	NKUA	EL
13	INAGRO, PROVINCIAAL EXTERNVERZELFSTANDIGD AGENTSCHAP IN PRIVAATRECHTELIJKE VORM VZW	InAgro	BE
14	UNIWERSYTET LODZKI	UL	PL
15	FUNDACION PARA LAS TECNOLOGIAS AUXILIARES DE LA AGRICULTURA	TEC	ES
16	DELPHY BV	Delphy	NL
17	INSTITUTO TECNOLOGICO DE ARAGON	ITAIN	ES
18	ZEMNIEKU SAEIMA	ZSA	LV
19	SOCIEDAD ARAGONESA DE GESTION AGROAMBIENTAL SL	SARGA	ES
20	AGROTIKOS KTINOTROFIKOS SYNETAIRISMOS KATOUNAS TO VIOLOGIKO AGROKTIMA	TBA	EL
21	SOCIETA ITALIANA DI VITICOLTURA ED ENOLOGIA	SIVE	IT
22	NILEAS-SYNETAIRISMOS PISTOPOIIMENON AGROTIKON PROIONTON DIMOU NESTOROS MESSINIAS	NILEAS	EL
23	CONSEIL DES VINS DE SAINT-EMILION	CVSE	FR
24	ASOCIATIA OPERATORILOR DIN AGRICULTURA ECOLOGICA BIO ROMANIA	BIORO	RO
25	RI.NOVA SOCIETA COOPERATIVA	RI.NO	IT
26	AGRO DIGITAL SOLUTIONS	AgroDS	LT
27	NATIONAL PAYING AGENCY	NPA	LT
28	AGENZIA PROVINCIALE PER I PAGAMENTIDELLA PROVINCIA AUTONOMA DI TRENTO	APPAG	IT
29	AGENTIA DE PLATI SI INTERVENTIE PENTRU AGRICULTURA	APIA	RO
30	QUEEN MARY UNIVERSITY OF LONDON	QMUL	UK

DISCLAIMER

Document History

Version	Date	Contributor(s)	Description
v0.1	18/10/2023	Y. Oikonomidis, I. Chrysakis (INTRA)	ToC
v0.2	20/12/2023	Y. Oikonomidis, I. Chrysakis (INTRA), N. Vesel (SINER), A. Turkmayali (IDSA), I. Ilie (SIEM), A. Retico, S. Sestili (ALMA), P. Ramírez, J. García, R. Lazcano (ATOS)	Content to INTRA sections, Contributions from partners to sections 3-6 and Annex I
v0.3	08/01/2024	Y. Oikonomidis, I. Chrysakis (INTRA), K. Railis, A. Lakka (SYN), K. Chandramouli (QMUL), S. Rizou, I. Sotiropoulos (SLG)	Merged contributions from partners to sections 3, 5, and 6.
v0.4	15/01/2024	Y. Oikonomidis, I. Chrysakis (INTRA), K. Railis, A. Lakka (SYN), S. Rizou, I. Sotiropoulos (SLG), I. Ilie (SIEM)	Merged input for sections 3 and 5.
v0.5	17/01/2024	Y. Oikonomidis, I. Chrysakis (INTRA), N. Vesel (SINER)	Merged final input, Conclusions, Ready for internal review.
v0.6	26/01/2024	Y. Oikonomidis, I. Chrysakis (INTRA), S. Rizou, I. Karvelas (SLG), K. Railis (SYN), A. Tudorascu, M. Angheloiu (SIMAVI)	Added subsection 8.3, Addressed Internal reviewers' comments, Prepared for final submission
v0.7	30/01/2024	Y. Oikonomidis, I. Chrysakis (INTRA)	Submission candidate
v1.0	31/01/2024	T. Zahariadis, G. Athanasiou, K. Railis (SYN)	Finalization

Document Reviewers

Date	Reviewer's name	Affiliation
26/01/2024	K. Railis	SYN
25/01/2024	S. Rizou, I. Karvelas	SLG

Table of Contents

Table of Figures

Table of Tables

Definitions, Acronyms and Abbreviations

AAA	Authentication, Authorization and Accounting
ABE	Attribute-Based Encryption
ACS	Access Control System
AI	Artificial Intelligence
AIPT	Aerial Image Processing Toolkit
API	Application Programming Interface
BYOC	Bring Your Own Collection
CA	Consortium Agreement
CI	Continuous Integration
CD	Continuous Delivery
DKM	Decentralised Knowledge Management
DL	Deep Learning
DLT	Distributer Ledger Technology
DMT	Data Model Translator
DOCG	Denomination of Controlled and Guaranteed Origin
DP-SGD	Differentially private Stochastic Gradient Descent
DRD	Drone Data Toolbox
DSS	Decision Support Systems
EO	Earth Observation
EOD	Satellite Earth Observation Data Capturing toolbox
FDML	Federated Deep Machine Learning
FedAvg	Federated Averaging
FL	Federated Learning
GAN	Generative Adversarial Network
GPS	Global Positioning System
GTP	Geo-tagged Photo
HFL	Hierarchical Federated Learning
HTTP	Hypertext Transfer Protocol
IDS	International Data Space
IOTD	IoT sensors Data Toolbox
ISQTB	International Software Testing Qualifications Board
JSON	JavaScript Object Notation
JWT	JSON Web Token
KPI	Key Performance Indicator

ML	Machine Learning
NFC	Near Field Communication
OGC	Open Geospatial Consortium
OIDC	OpenID Connect
OS	Operating System
OSM	Open Street Maps
PATE	Private Aggregation of Teacher Ensemble
PVC	Persistent Volume Claim
QR	Quick Response
REST	Representational State Transfer
SAML	Security Assertion Markup Language
SCM	Source Code Management
SSL	Secure Sockets Layer
STAC	Spatio-Temporal Asset Catalogs
TDD	Test-driven Development
TLS	Transport Layer Security
UML	Unified Modeling Language

Executive Summary

This document describes the first version of the AgriDataValue Platform, which combines the features from the initial releases of the AgriDataValue components created under Work Packages (WP) WP2, WP3, and WP4.

The AgriDataValue integrated platform's initial version offers sub-system level integration, primarily concentrating on the functionality of the individual components as well as the integrated platform as a whole. It provides an overview of the infrastructure in use and provides details on various aspects of integration and deployment. In fact, the following topics are clarified by this report:

- The infrastructure and integration framework utilized in the project's technology development pipelines;
- The most recent developments in the AgriDataValue component development;
- The deployment and configuration perspective of the AgriDataValue Platform;
- The methodology for the verification and validation of the integrated platform and its constituent parts.

Deliverable D2.2, " AgriDataSpace Platform of Platforms V1," is anticipated to contain the second iteration of the AgriDataValue Platform at M24.

1 Introduction

This deliverable aims to give an overview of the first version of the AgriDataValue Platform and its components, as well as details the project's integration and validation tasks.

This document discusses the AgriDataValue Integrated Platform, taking into account both the platform's functionality and the integration of its component parts. Updates and developments related to the components that make up the integrated platform, revisions to the integration specifications and tools, and activities related to the platform's verification and validation are the main focus of the first integrated prototype.

1.1 Scope and purpose

The scope and purpose of the document are introduced in this section. D2.1, the first of three deliverables for the AgriDataValue platform, is presented in this document. The initial release of the AgriDataValue components integrated into the AgriDataValue platform is provided by this deliverable which was aided by the technical WPs and the owners of each component.

The deliverable's objectives are to document:

- the AgriDataValue Platform components' functionality and design;
- how the various parts come together to form a cohesive framework that provides coordinated services;
- how AgriDataValue‘s deployment options support the platform's reproducibility and uphold its resilience and performance capacity through design; and
- how the platform will be verified and validated.

1.2 Document overview

The remainder of this document is comprised of the following sections:

- Section 2 presents the first version of the AgriDataValue platform, the steps that were taken in order to ensure graceful cooperation among the various AgriDataValue subparts and the cloud-native considerations that led to the adoption of Kubernetes[1] as a deployment infrastructure and container orchestration framework.
- Section 3 presents the components group Decentralised data capture management & in-situ pre-processing tools (from WP3 and WP4)
- Section 4 presents the components group Edge Cloud Analytics Suite (from WP2)
- Section 5 presents the components group Data Security, Privacy, Traceability & Sharing (from WP2)
- Section 6 presents the component group AI-based Cloud platform (from WP2)
- Section 7 describes the infrastructure and tools that have been setup, deployed, configured, and how they are leveraged to assist in integrating and deploying an AgriDataValue component.

[1] https://kubernetes.io

- Section 8 includes the verification and validation plan, describes the process that is being followed for the update of the technical requirements, and summarizes the Verification and Validation reports collected so far for the various AgriDataValue components.
- Section 9 provides the conclusions and next steps.
- Annex I provides:
    - the Message bus data schema
    - the template used to represent the general information of the verification and validation of any AgriDataValue component.
    - the Message bus data schema
    - the EOD component API description
    - the Fleviden Framework

As far as the reader is concerned, given that this deliverable documents the initial version of the integrated ADV platform and its sub-components, it is suggested to be read in a linear manner.

2 AgriDataValue Integrated Platform

This section aims to provide an overview of the integrated AgriDataValue platform prototype, document issues raised regarding the ease of use of the system during development and implementation, and present and evaluate the procedures of the framework that was created for the automated configuration and deployment of the entire AgriDataValue platform.

2.1 AgriDataValue platform as a whole

The AgriDataValue architecture and a synopsis of the platform's components are presented in D1.3, the first step towards the development of the AgriDataValue integrated platform. To keep this deliverable self-contained, we provide an overview of the AgriDataValue platform architecture in Figure 1. Sections 3-6 of this document provide a more thorough analysis and expansion of the platform's components. This section examines our design decisions and the addition of automated flows, with a primary focus on the deployment of the AgriDataValue platform.

Our efforts have been focused on developing a reliable method for easily and reliably deploying, updating, and maintaining the various parts of the integrated AgriDataValue platform due to the significance of the project and its potential results. Because of this, a cloud-native strategy was chosen, along with a reliable toolkit for contemporary architectures that included Docker and Kubernetes. This toolkit is being adopted in a way that places Kubernetes at the centre of the AgriDataValue architecture. The following illustrates how the components' development is directly impacted by the chosen approach.

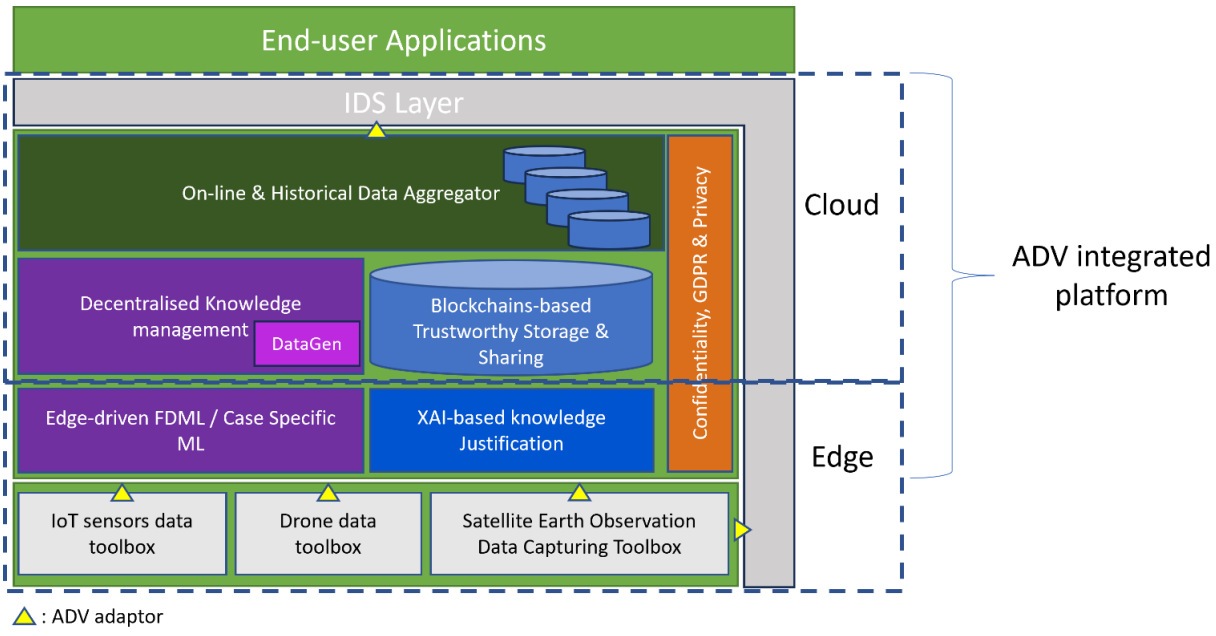


Figure 1: High-level reference architecture overview

The AgriDataValue elements that will be incorporated into the platform's initial release are the following:

- FDML (Edge Cloud Analytics Suite)
- XAI (Edge Cloud Analytics Suite)
- SECURESTORE (Data Security, Privacy, Traceability & Sharing)
- CHAINTRACK (Data Security, Privacy, Traceability & Sharing)
- ACS (Data Security, Privacy, Traceability & Sharing)

- DKM (AI-Based Cloud Platform)
- DATAGEN (AI-Based Cloud Platform)

The components above were prioritized as they form the core of the ADV platform, and also, they were somewhat detached from the pilots in the sense that their initial development could proceed and reach a suitable maturity level without blocking dependencies on the Use cases or User scenarios. Of course, from that point forward, pilot and end-user related information is important.

The rest of the components are currently under development and will be integrated into the platform in the upcoming period. However, there are already plans for their smooth integration and preliminary integration tests have already been performed for some of these components.

2.2 AgriDataValue platform integration considerations

AgriDataValue and its component parts are inherently complex, so during the development of the AgriDataValue integrated platform, factors like the requirement for dependable and simple component installation acted as major motivators. The introduction of the cloud-native Kubernetes approach also brings up several integration-related considerations that must be addressed to ensure that an application is easy to install and maintain over time, as well as robustness and integrity during deployment operations. Kubernetes is an open-source container orchestration platform which makes software deployment, scaling, and management automated.

Initially, before deploying an application as a container (or group of containers) to Kubernetes, a set of YAML files must be created. These YAML files describe various components of the deployment, exposed service (if any), employed storage, config maps, and secrets with important deployment information (e.g., values for environment variables), and jobs for necessary initialization and other recurring tasks. The deployment of our custom applications requires these files. For our applications as well as external components like Keycloak[2] (core of ACS component) and Apache Kafka[3] which are further described in section 7, we employ an automated configuration and deployment flow that necessitates manual file authoring; this process is covered in the sections that follow.

Since the noted factors have been suitably taken into account, an automated workflow for the straightforward and default configuration and deployment of the AgriDataValue platform has been developed.

2.3 Configuring and deploying the AgriDataValue platform

The subsections that follow describe some technical aspects of the ADV platform with respect to its deployment and configuration, having Docker and Kubernetes at the core of this process. With the use of Docker, applications can be automatically deployed into lightweight containers, enabling them to function effectively in separate environments.

[2] https://www.keycloak.org/: Keycloak is an open-source software solution designed for contemporary applications and services that enables single sign-on with identity and access control. It offers features including user management, two-factor authentication, permissions and roles management, creating token services, and support for multiple protocols including OpenID, OAuth version 2.0, and SAML.

[3] https://kafka.apache.org/: Apache Kafka is a platform for stream processing and distributed event storing. It offers a single, low-latency, high-throughput platform for managing real-time data feeds.

2.3.1 Docker & Registry

2.3.1.1 Dockerfile

Ideally, the minimum requirement for the smooth deployment and integration of every AgriDataValue component is a Dockerfile. Ideally, the Docker image created by the Dockerfile provided by the developer should run without with a simple docker run command. That is the case when the ENTRYPOINT and CMD arguments are used correctly, along with the ports to EXPOSE. Of course, for a proper run various environmental variables have to be set.

2.3.1.2 Registry

The container registry for the AgriDataValue project runs at https://registry.git.agridatavalue.eu.

Let's consider an image for the present project. The project lies under the AgriDataValue / Guidelines subgroup. This specific image will be accessible under registry.git.agridatavalue.eu/agridatavalue/guidelines/kubernetes. If an image has been built with a vX.X.X tag, then the user can pull the image via registry.git.agridatavalue.eu/ agridatavalue/ guidelines/ kubernetes:vX.X.X.

Generally, a registered user can push and pull images to the registry with their GitLab credentials, as long as they are a member of the repository under discussion.

The example GitLab CIs have been configured to automatically build the essential Docker images. As long as a project includes an appropriately modified (if needed) .gitlab-ci.yml at the root of the repository the images will be built.

2.3.2 Kubernetes essentials

In the following brief explanations of Kubernetes essential objects for the deployment of an application are provided.

2.3.2.1 ConfigMap

A ConfigMap is an API object used to store non-confidential data in key-value pairs. Pods can consume ConfigMaps as environment variables, command-line arguments, or as configuration files in a volume. More can be found at https://kubernetes.io/docs/concepts/configuration/configmap/.

An example ConfigMap is shown below:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: go-example-app
  namespace: agridatavalue
data:
  example: example-value
```

2.3.2.2 Secret

A Secret is an object that contains a small amount of sensitive data such as a password, a token, or a key. Such information might otherwise be put in a Pod specification or in a container image. Using a Secret means that you don't need to include confidential data in your application code. See more: https://kubernetes.io/docs/concepts/configuration/secret/.

The values stored in a Kubernetes Secret must be Base64 encoded. For instance, let's assume that we need to create a Secret entry titled `example` with value `examplesecret`. We should run the following command:

```
$ echo -n "examplesecret" | base64
ZXhhbXBsZXNlY3JldA==
```

Then a Secret is created as follows:

```
apiVersion: v1
kind: Secret
metadata:
  name: go-example-app
  namespace: agridatavalue
type: Opaque
data:
  example: ZXhhbXBsZXNlY3JldA==
```

2.3.2.3 Persistent Volume Claim

A PersistentVolumeClaim (PVC) is a request for storage by a user. It is similar to a Pod. Pods consume node resources and PVCs consume PV resources. Pods can request specific levels of resources (CPU and Memory). Claims can request specific size and access modes (e.g., they can be mounted as ReadWriteOnce, ReadOnlyMany or ReadWriteMany, see the field AccessModes[4]).

More can be found at https://kubernetes.io/docs/concepts/storage/persistent-volumes/.

An example PersistentVolumeClaim can be seen below:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: go-example-app
  namespace: agridatavalue
  labels:
    type: local
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 100Mi
```

The above PersistentVolumeClaim will requests 100MiB storage from the cluster, thus creating a PersistentVolume.

2.3.2.4 Deployment

A Deployment is essentially a description of a desired state, and the Deployment Controller changes the actual state to the desired state at a controlled rate. Deployments can be defined to create new ReplicaSets, or to remove existing Deployments and adopt all their resources with new Deployments. See more: https://kubernetes.io/docs/concepts/workloads/controllers/deployment/.

An example Deployment can be seen below:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: go-example-app
  namespace: agridatavalue
  labels:
    app: go-example-app
spec:
```

[4] https://kubernetes.io/docs/concepts/storage/persistent-volumes/#access-modes

```
  replicas: 1
  strategy:
    rollingUpdate:
      maxUnavailable: 1
    type: RollingUpdate
  selector:
    matchLabels:
      app: go-example-app
  template:
    metadata:
      labels:
        app: go-example-app
    spec:
      imagePullSecrets:
      - name: agridatavalue-regcred
      containers:
      - name: go-example-app
        image: REGISTRY_IMAGE:REGISTRY_IMAGE_TAG
        imagePullPolicy: Always
        ports:
        - name: go-app-port
          containerPort: 8080
        env:
          - name: EXAMPLE_SECRET
            valueFrom:
              secretKeyRef:
                name: go-example-app
                key: example
          - name: EXAMPLE_CONFIG
            valueFrom:
              configMapKeyRef:
                name: go-example-app
                key: example
        volumeMounts:
          - name: go-app-volume
            mountPath: /opt/app/data
      volumes:
        - name: go-app-volume
          persistentVolumeClaim:
            claimName: go-example-app
```

You may notice that a typical Deployment combines the previous elements. It defines imagePullSecrets, containers and volumes, among other things. Various volumeMounts can be configured for the container. Additionally, an env segment defines essential environment variables, either directly or via values received from ConfigMap objects and Secret objects, while a port segment defines the ports of the container under discussion. Please refer to the Deployment's documentation for further information.

2.3.2.5 Service

A Service is a method for exposing a network application that is running as one or more Pods in a K8s cluster. See more: https://kubernetes.io/docs/concepts/services-networking/service/.

An example Service can be seen below:

```
apiVersion: v1
kind: Service
metadata:
  name: go-example-app
  namespace: agridatavalue
```

```
spec:
  selector:
    app: go-example-app
  ports:
  - port: 8080
    targetPort: go-app-port
    protocol: TCP
  type: NodePort
```

This Service exposes the targetPort, i.e. the port defined as go-app-port in the Deployment to service port 8080. The type of this specific Service is NodePort which means it will expose the container's port to the port of the Kubernetes node.

2.3.3 Deployment platform

A Kubernetes cluster hosted by SYN will host the deployment of ADV components and their subcomponents. Within this cluster, a specific ADV namespace has been established. The ADV namespace's current deployments are displayed in Figure 2. Unless other needs arise during the project's duration (e.g., like the need for an additional namespace or another type of deployment practice), the majority of ADV's components will be deployed with the setup that is specified. The intercommunication and interoperability of ADV's subcomponents will be possible through Kubernetes' standard visual network abstraction layer in conjunction with the relevant Kubernetes services for each component, even if more than one namespace is required.

```
❯ kubectl -n agridatavalue get pods -o wide
NAME                                READY   STATUS    RESTARTS      AGE    IP             NODE             NOMINATED NODE   READINESS GATES
gitlab-agent-v1-8586bc578c-mdcps    1/1     Running   1 (3d ago)    3d     10.244.6.56    compute-r540-1   <none>           <none>
kafka-broker-0                      1/1     Running   3 (3d ago)    3d1h   10.244.5.93    compute-r540-0   <none>           <none>
kafka-controller-0                  1/1     Running   4 (3d ago)    3d1h   10.244.6.237   compute-r540-1   <none>           <none>
kafka-kafka-ui-848879fd46-9qjc6     1/1     Running   1 (3d ago)    3d     10.244.6.181   compute-r540-1   <none>           <none>
keycloak-0                          1/1     Running   9             3d1h   10.244.5.72    compute-r540-0   <none>           <none>
keycloak-postgresql-0               1/1     Running   2 (3d ago)    3d1h   10.244.5.82    compute-r540-0   <none>           <none>
```

Figure 2: An instance of the AgriDataValue namespace

Given the importance of data in the ADV context and the cloud-native paradigm that the project is implementing, cloud-native storage is a must. Based on this, a software storage cluster called Ceph[5] and the open-source Rook framework[6] have been used to configure storage. This combination allows for transparent and easy implementation of effective and configurable storage duplication for the end-user application, without requiring adherence to specific component design practices.

```
❯ kubectl get storageclasses -o wide
NAME                   PROVISIONER                     RECLAIMPOLICY   VOLUMEBINDINGMODE   ALLOWVOLUMEEXPANSION   AGE
rook-ceph-block (default)   rook-ceph.rbd.csi.ceph.com      Delete          Immediate           true                   2y170d
rook-ceph-block-ssd         rook-ceph.rbd.csi.ceph.com      Delete          Immediate           true                   2y100d
rook-ceph-bucket            rook-ceph.ceph.rook.io/bucket   Delete          Immediate           false                  2y163d
rook-cephfs                 rook-ceph.cephfs.csi.ceph.com   Delete          Immediate           true                   2y169d
```

Figure 3: Storage classes of the Kubernetes cluster

As a final note, docker-compose versions of the deployments will be made available to the component developers, as Kubernetes might not be accessible to all developers during the coding and debugging procedures. This will enable the local development process to be simplified and brought closer to the real environment, which is made possible using containers for both local testing and development.

[5] https://ceph.io/en/
[6] https://rook.io/

3 Decentralised data capture management & in-situ pre-processing tools

This chapter focuses on Decentralized data capture management and in-situ pre-processing tools to be implemented in the context of ADV. This collection of tools is going to offer capturing and pre-processing of data at the edge, leveraging IoT advancements in communications, computational power, and storage, as well as edge computing. This approach aligns with the in-situ processing paradigm, via the distribution of processing load and reduction of the amount of transferred data. At the time of this deliverable's authoring, the identified toolboxes include:

- the IoT sensors data toolbox (IOTD),
- the Terrestrial Geotagged-Photos (GTP App),
- the Drone Data toolbox (DRD), and
- the Earth Observation Data capturing toolbox (EOD)

These discrete toolboxes, that serve as the main data sources of ADV, comprise a multi-modal data collection and in-situ pre-processing mechanism that will achieve a complete and accurate representation of knowledge, to serve ADV's mission and long-term vision. In the following, more thorough descriptions of the comprising toolboxes will be presented.

3.1 IoT sensors data toolbox (IOTD)

The IoT sensors data toolbox (IOTD) is the toolbox that focuses on the in-situ monitoring, collection, and pre-processing of measurements originating in a series of IoT devices and sensors, deployed on the field. The IOTD toolbox is implemented in the context of *Task 3.2: Smart Farming Support and CAP compliance toolbox*. The IOTD toolbox consists of following three main sub-toolboxes:

- Open-Field Crops' IoT Sensors Data Capturing Toolbox
- Greenhouse / Farm Air Quality & Animals' Wearable IoT Data Capturing Toolbox, and
- Terrestrial Geotagged-Photos' Data Capturing Toolbox

The first two toolboxes focus on the collection and pre-processing of measurements originating from sensors and IoT devices that are deployed on the field, while the third one is focused on the capturing of geotagged photos via a hand-held device (i.e., a smartphone or tablet) on the field. In the following paragraphs, more details about the IOTD toolbox as-a-whole, as well as its sub-toolboxes, are provided.

3.1.1 Description

In this paragraph a description of main operation principles for the IOTD toolbox is provided. As stated, the IOTD toolbox serves as one of the main data sources of ADV, as it enables the real-time monitoring and collection of measurements originating in a variety of IoT devices and sensors that are deployed on the field for a series of measurable entities. Moreover, it supports visual information of crops' status that is, combined with in-situ measurements.

The **Open-Field Crops' IoT Sensors Data Capturing Toolbox** will be based on SynField™, SYN's flexible sensing and precision agriculture solution [1]. A SynField™ device is depicted in **Error! Reference source not found.**. For the Open-Field Crops' IoT Sensors Data Capturing Toolbox the measured entities include, but are not limited to:

- Rain accumulation
- Wind speed & direction
- Air temperature and relative humidity
- Soil temperature, humidity, and electrical conductivity
- Leaf-wetness
- Solar radiation

Figure 4: A SynField™ device

In addition to this, third-party platforms that are currently utilized in pilot sites will be integrated, via their APIs (if available and open) to form a complete solution. With respect to integration of potentially utilized third-party platforms, priority will be given to the ones that come with an openly accessible (within the scope of the project), and well-documented API.

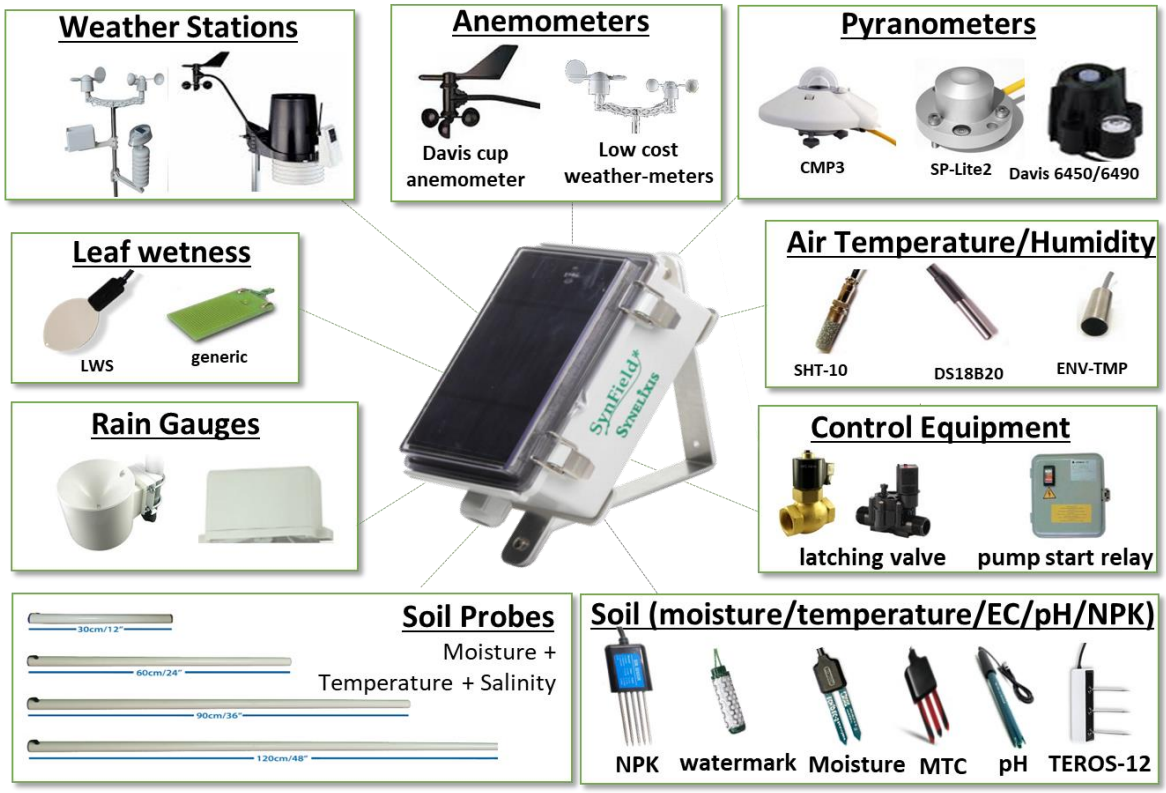


Figure 5: A SynAir™ Installation

As far as the **Greenhouse / Farm Air Quality & Animal Wearable IoT Data Capturing Toolbox** is concerned, the Greenhouse / Farm Air Quality part of the toolbox will be based on SYN's SynAir™ [2], a versatile sensor platform that can support a wide range of air quality sensors, and an extension to SynField™. A SynAir™ installation is depicted in Figure 6. Like the previous toolbox, if/when existing air quality sensors and/or historical data are available, they will be integrated into ADV via the APIs they offer. For the Greenhouse/Farm Air Quality the measured entities include, but are not limited to:

- Temperature
- Relative humidity
- Barometric pressure
- Suspended particles
- Volatile Organic Compounds (VOC)

- Various gases (e.g. CO, CO_2, NH_3)

On top of this, with respect to Animals' Wearable IoT devices, priority will be given to solutions that are currently utilised by the pilots. In case the currently utilised solutions are not appropriate for integration within the ADV platform, off-the-shelf smart collars for cows and/or pigs will be utilised, with a preference for solutions that offer openly accessible, well-documented and ready-to-integrate APIs.

Figure 6: A SynAir™ Installation

The operational principles of the previous sub-toolboxes are identical. IoT devices and sensors are deployed on the field, whether this is an open-field, greenhouse, farm or a stable. The deployed sensors are connected either wired or wirelessly to a gateway that collects their measurements, performs an appropriate pre-processing at the edge (i.e., at the gateway), and stores (publishes) them to another remote location for further processing if/when needed by other components of the ADV platform. The recorded measurements are available, accessible and can be visualised on a per-user basis via a relevant user interface (UI).

Finally, the **Terrestrial Geotagged-Photos' Data Capturing Toolbox**, offers a visual indication of crops' state and will serve as input to plant disease prediction and prevention procedures that will be developed within ADV. From the end-user aspect, this toolbox will be realized as a hand-held device, i.e. smartphone or tablet, offering the user the ability to capture photos of plants' leaves or other aspects.

It should now be clear to the reader, that the multi-modal data collection will offer a complete overview of the monitored entity's status.

3.1.2 Development view

3.1.2.1 Component diagram

Figure 7 depicts the component diagram of the IOTD toolbox. In this diagram, the sub-components of the IOTD toolbox are presented, and their internal, as well as external interactions are captured. The IOTD toolbox is comprised of an IoT sensors Northbound API, which is utilized by IoT sensors to interact with the SynField™ and SynAir™ devices. SynField forwards the collected measurements to the ADV Data Model Translator, which is also the point-of-entry for third-party platforms that will be integrated within ADV. The translated measurements are then going to be either published to other ADV components (IDS, STORE, Kafka Message Bus) or be accessible via an appropriate API. This is the job of the ADV-adapted data publisher & API sub-component. Finally, leveraging the aforementioned API, an IOTD User Interface will be available to the end-users of the toolbox/ADV platform.

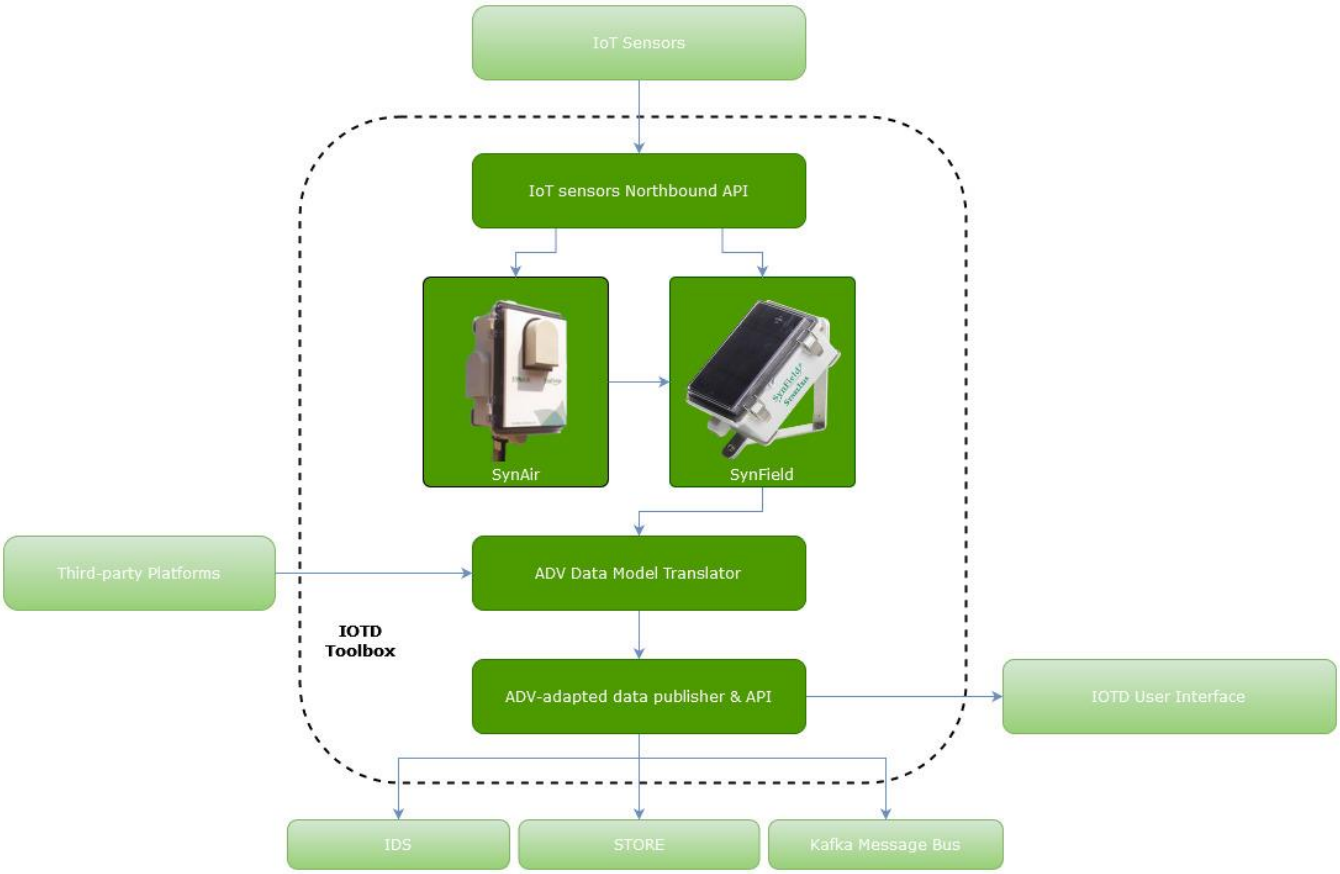


Figure 7: Component diagram for the IOTD Toolbox

3.1.2.2 Building blocks

3.1.2.2.1 IoT sensors Northbound API

The IoT sensors Northbound API is the IOTD component via which the IoT sensors interact with the toolbox in order to record their collected measurements. The IoT sensors Northbound API is a low-level API consisting of multiple device drivers to support the communication with sensors, management, and data collection. A deep understanding of these low-level interactions between sensors and the actual SynField™ nodes is considered out of scope for this deliverable. However, we should note that the IoT sensors Northbound API is implemented in its entirety by SynField™. In case of third-party platform utilisation, the implementation or adaptation of any available third-party APIs is not within the scope of the project, unless an owner/contributor of the third-party platform is willing to make an adaptation that is considered necessary. Nevertheless, in case the supported collection of sensors is not considered adequate and new needs occur within the duration of the project, the IoT sensors Northbound API will be expanded to support these additional sensors.

3.1.2.2.2 ADV Data model translator

The ADV Data model translator (DMT), or simply ADV adapter, is a subcomponent of IOTD that focuses on the translation of measurements received via SynField™, SynAir™, or a third-party platform, to the ADV data model. DMT is considered a vital part of the IOTD toolbox, and the ADV overall because it serves as an enabler for the interoperability between, not only the components of ADV, but also the numerous heterogeneous devices that are going to be integrated within the toolbox. For the third-party platforms that will be integrated within the ADV, the ADV Data model translator will be the point of entry.

In the current version of the toolboxes, the initial focus of this sub-component is to map the toolbox entities, like sensors, measurements, fields, parcels, etc., to the Agricultural Information Model (AIM), before proceeding to the integration of the IDS part of the ADV data model. It should be noted that priority has been given to the translation / adaptation of the SynField™ measurements to the ADV data model before the APIs of other third-party platforms follow.

3.1.2.2.3 ADV-adapted data publisher & API

The ADV-adapted data publisher is a subcomponent of IOTD that serves as a publisher for the collected measurements. It is responsible for communicating the collected and pre-processed measurements to other ADV components, as needed. The publisher is going to support the saving of measurements to the STORE component, their communication to IDS connector, and, of course, their publishing to the central Kafka bus of the ADV platform. Furthermore, the ADV-adapted data publisher is going to offer an API for the retrieval of measurements on per-user basis.

3.1.2.2.4 IOTD User Interface (UI)

The IOTD User Interface (UI) is going to serve as the user interface for the toolbox, a place where the end-users will be able to visualize all the collected measurements for their fields, greenhouses, farms, stables, or any place where a sensor or IoT device is deployed. The end-users will be able to view their fields or areas of interest, according to their sensor installations. On top of this, photos that are captured via the Terrestrial Geotagged-Photos Data Capturing Toolbox will be visualized via both the relevant mobile application, and possibly via the IOTD UI.

3.1.3 Process view

3.1.3.1 Sequence diagram

Figure 8 presents a simplified sequence diagram for the IOTD component that depicts the flow of information within the toolbox. Raw data initially received from the deployed sensors and IoT devices arrive to SynAir and SynField nodes via the IoT sensors Northbound API. SynField gathers measurements concerning air, soil, wind, and rain directly from the Northbound API, as well as measurements concerning air-quality from SynAir. The processed measurements are then forwarded to the ADV Data Model translator, together with measurements originating in potentially integrated third-party platforms. These processed measurements are then made available to the ADV-adapted data publisher and are accessible on a per-user basis via the exposed web API. Finally, the processed and translated measurements are available for the end-user via the IOTD UI.

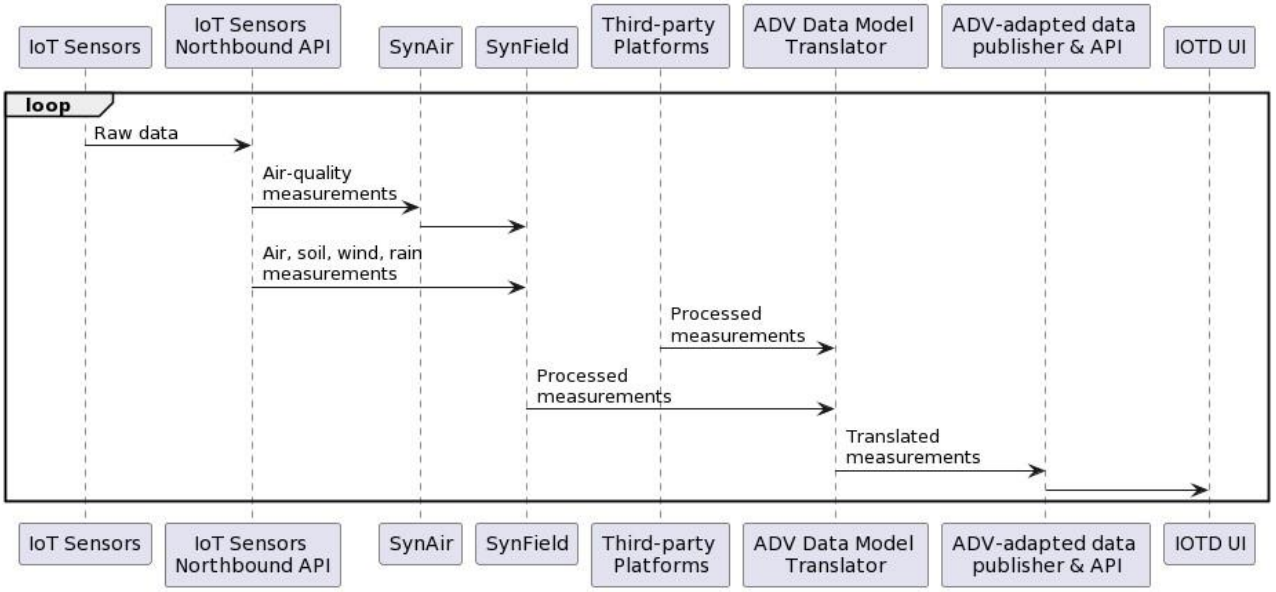


Figure 8: Simplified sequence diagram for the IOTD toolbox

3.1.4 Interfaces

3.1.4.1 Data models used in interfaces

As already mentioned in previous paragraphs, the adopted ADV data model will be a combination of the Agricultural Information Model (AIM) and other IDS essential components. At the current phase of the project, the mapping between measurements and AIM entities has been the focus concerning the ADV data model. Since the components of IOTD are in a state of intensive development, the current version of the mapping is not considered stable, and it is preferred to not be recorded in this deliverable. This mapping is currently in progress and the plan is to be completed in the following month(s).

3.1.4.2 Description of APIs

Like the ADV data model and interfaces, the APIs and Kafka message formats for IOTD are not currently stable and will not be recorded in the present document. However, deliverable D3.2, which will focus on the IOTD toolbox, is going to include detailed descriptions of the IOTD toolbox and its sub-components (sub-toolboxes).

3.1.5 Technologies and implementation details

In this paragraph, the employed technologies are briefly described. To begin with, the IoT sensors Northbound API, already implemented and inherent in the SynField™ hardware platform is a low-level API, implemented in C/C++ and the specifics of its implementation are out-of-scope of this project.

The ADV Data model translator will be implemented in Python[7], leveraging Django[8] and Celery[9] for asynchronous task execution.

[7] https://www.python.org/
[8] https://www.djangoproject.com/
[9] https://docs.celeryq.dev/en/stable/

The ADV-adapted data publisher & API will be implemented in Python as well, leveraging PostgreSQL and PostGIS[10], Celery, MinIO[11] and GDAL[12], and it will be able to interface with Kafka, MinIO and PostgreSQL for the storage/publishing of measurements. The API part of this component will be implemented in Django and Django REST Framework[13], and it will be served via a combination of Nginx[14] and Gunicorn[15]. Swagger[16] documentation of the API will be available for smoother integration with other ADV components that might need to access it.

Finally, the IOTD UI component, though its development has not yet launched, will be implemented in either React[17] or Vue.js[18], leveraging a collection of built-ins and tools.

Since the IOTD toolbox is still under development, a complete list of the utilised technologies will be offered in deliverable D3.2, which will present the first version of the Smart Farming Support and CAP compliance toolbox.

3.2 Terrestrial Geotagged-Photos (GTP App) Data Capturing Toolbox

The agricultural industry and in particular the farming community has constantly faced the threat of pests and environmental disruptions and is being considered a severe threat for food security and economic stability for both farmers and general public [3]. Traditionally, such challenges are addressed through the local knowledge of farmers which has been passed down through generations and has paved the way for mitigating some of the impact of pests. While the use of advanced scientific tools and solutions have been largely adopted by various industrial sectors in the European region, the use of mobile computation and cloud deployment of deep-learning network models for the automation of agricultural services has not been fully exploited. Among the several types of agricultural plants which are affected by pests, infestation of leaves is regarded to have the maximum impact upon the food production. To address this challenge, the Terrestrial Geotagged-Photos Capturing Toolbox (GTP App, the commercial reference to be adopted within this chapter and within the project) is designed to establish a seamless process to be adopted by farmers to keep track of the crop health.

3.2.1 Description

The GTP App is built on the premise that following the pervasive use of mobile technologies, the farmers could be equipped with a simple application that would allow them to capture and upload the relevant visual information gathered from the field and assess the quality of the crop along with the identification of any potential pest outbreak. The pictures captured by the application is geotagged for marking the spatial location using Global Positioning System (GPS) information often associated with the latitude and longitude measures. The novelty of the proposed application is the ability to function when operated under uncontrolled environment and requires no additional intervention from the farmers to complete the processing. The

[10] https://postgis.net/
[11] https://min.io/
[12] https://gdal.org/index.html
[13] https://www.django-rest-framework.org/
[14] https://www.nginx.com/
[15] https://gunicorn.org/
[16] https://swagger.io/
[17] https://react.dev/
[18] https://vuejs.org/

information gathered by the farmers is subsequently processed in the cloud and the result is then passed on to the farmer with additional domain knowledge on the health state of the crop or with additional information about the nature of the pest that is infecting the crop quality.

3.2.2 Development view

3.2.2.1 Component diagram

The component diagram of the GTP App is presented in Figure 9, which represents two components under development. The GTP application side depicts the end-user interface that is presented to the farmers with different user screens being accessible to the farmers. The component indicating the user interface, is presented to the farmers and includes simple interaction guidelines. The information presented in the application has been validated for the user-centric approach that has been adopted and provides a user-friendly transition from screen to screen. The design choices adopted in the application offers an intuitive user interaction and requires no further training or support to use the application. The application also integrates several technical components that are hidden from the user. The use of the local storage component allows the application to cache most relevant information to be presented to the users. The image capture model interfaces with the camera API available within the mobile operating system (OS) to enable image capture. The geo-location service allows for the users to tag the location of the image that is being captured. Additionally, the communication module enables data exchange with the backend services referred to as GTP server side.

The GTP server side integrates the user authentication and authorisation services and furthermore integrates the complex computation of image analytics and AI algorithms that provide classification outcomes to represent the health of the crop under consideration. Additionally, data aggregation and data annotation components are integrated as two key backend services. A detailed description of these components is presented in the next section.

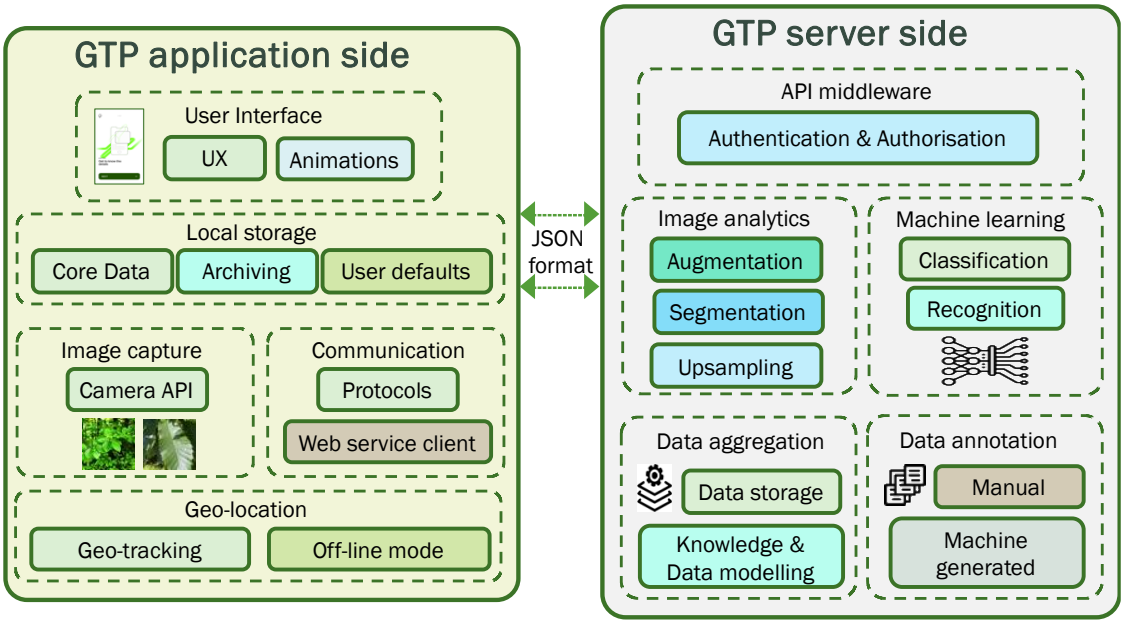


Figure 9 - GTP Application with the front end and backend processing

3.2.2.2 Building blocks

3.2.2.2.1 Image capture

The image capture module directly leverages the functionality of the underlying operating system of the mobile platform and allows the farmers to capture the images of the leaf under consideration. The camera screen navigation is directly enabled from the main screen of the GTP application.

3.2.2.2.2 Geo-location

The geo-location services interface the Open Street Map (OSM) like services to allow the farmers to assign and tag the latitude and longitude measures associated to the region. The information once assigned to the image that has been captured will be stored in the local storage and bundled into a single package of information that will be transmitted to the backend services.

3.2.2.2.3 Communication protocols

The communication protocols module integrates multi-stack services that allows the data exchange and communication to take place from the mobile application to the backend services. The communication API seamlessly integrates with the available communication mode of operation such as WiFi and mobile network. It is also noted that in case if there is no network coverage available across the farming field, the image captured and packaged with the GPS coordinate will be stored in the cache and once the network is re-established the cached information will be posted to the backend system, in background mode of operation that requires no intervention from the users.

3.2.2.2.4 Image analytics

Once the image collected have successfully been posted to the backend server for postprocessing, a data handling pipeline is initiated, in which the algorithms for augmentation, segmentation, super-resolution are applied to improve the overall quality of the images that have been gathered from the field. The sequence of algorithms that have been applied enables the processing of images collected from uncontrolled environment that is most suited for farmers to capture the data from the field.

3.2.2.2.5 Machine learning

The pre-processed images from the image analytics pipeline are then subsequently subjected to the classification model that integrates classes of pest. The network will be trained with the most commonly occurring pest categories that affect the leaf. The outcome from the machine learning algorithm will be used to generate a report on the status of the leaf, with the domain knowledge gathered from experts that will be sent back to the farmers as a report.

3.2.2.2.6 Data aggregation

This component represents the necessary of information that is needed to be gathered on the knowledge of pest categories, their impact on the leaf and the impact on the neighbouring trees as pests often do tend to spread from one crop to another. The knowledge model used will incorporate the information that is needed to be presented to the farmer to enrich the information about the state of the crop. Additionally, the knowledge needed for the farmer to take preventive and mitigation steps as required to protect crop quality.

3.2.2.2.7 Data annotation

The component forms one of the core elements of the GTP App, which holds the annotated data representing the pests. The GTP app will adopt two forms of data annotation processes, namely manual process and machine-generated. The annotated results that are generated from the machine learning algorithm will be continually monitored and managed to improve the quality of the training and by automating the process of annotation using pre-trained models.

3.2.3 Process view

3.2.3.1 Sequence diagram

The sequence diagram of interactions that take place between the different components of the application is presented in Figure 10.

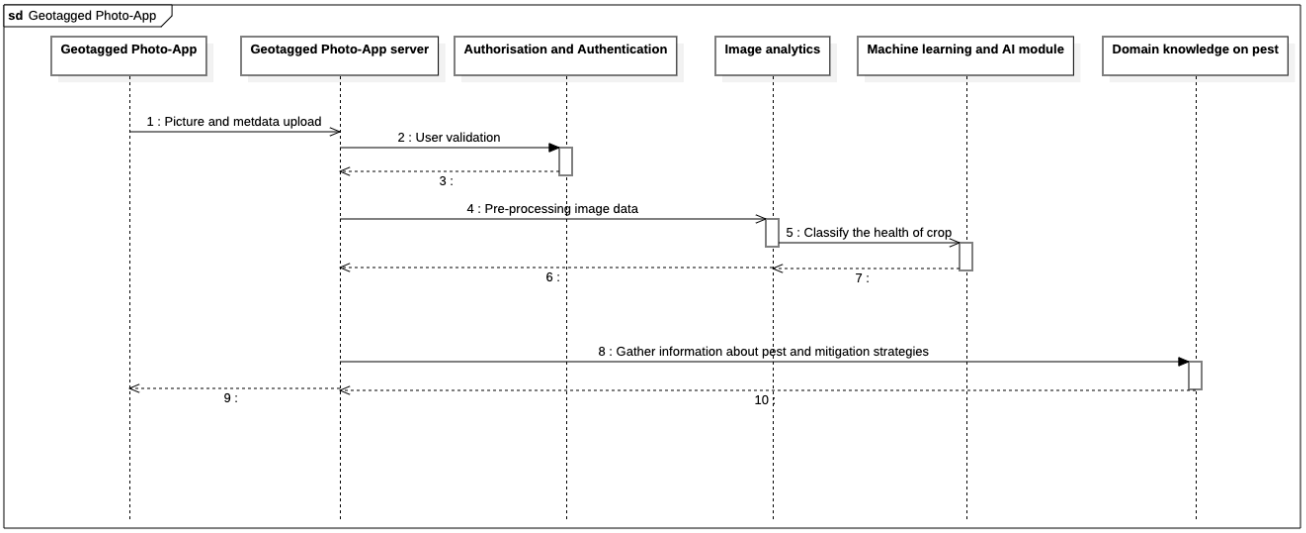


Figure 10 - Geotagged Photo-App sequence diagram

3.2.4 Interfaces

3.2.4.1 Data models used in interfaces

The GTP App will extend the data model adopted in ADV, which has been derived from AIM ontology specification. As this information space is less populated with the list of concepts related to the pests, the knowledge gained in ADV will be subsequently added to the data model. Additionally, JSON-based representation of data will be adopted for the integration of the component and services that were outlined in the earlier section.

3.2.4.2 Description of APIs

It is important to note that the GTP App distribution is aimed at multiple front-end users (farmers) while the deployment of the server will be aimed at handling multiple requests that are being received from farmers. Therefore, the communication protocols implemented in the GTP App front end will adopt a stateless implementation and use RESTFul API services to exchange information back to the server. Similarly, the frontend will also implement an asynchronous protocols service to receive information on the pest categories after processing from the backend.

3.2.4.3 Technologies and implementation details

The GTP App will use the latest release of software stack including support extended for the Android platform for the front end, with Python and Java services being used for the backend services. The backend system will also implement necessary interfaces required to handle large volume of data that is generated and shared with the backend services to be processed and analysed.

3.3 Drone data toolbox (DRD)

The Drone Data Toolbox (DRD) is one of the main data sources of ADV, along with IOTD and EOD toolboxes. As a data source it can be considered complementary to the EOD toolbox, rather than the IOTD toolbox, which

interfaces with lower-level entities, such as IoT sensors and other devices. The main mission of the DRD toolbox is to support the capturing and post-processing of aerial geotagged photos. As such, the drone and the utilized camera are considered a vital part of the toolbox with respect to data capturing, while the processing of the captured images, when combined with data originating in EOD and IOTD toolboxes will offer a complete understanding of the monitored entities' (i.e., fields) state.

3.3.1 Description

DRD implements what is described as Aerial Geotagged-Photos' Data Capturing Toolbox in the context of Task 4.2 *Climate Monitoring and CAP compliance toolbox*. In this context, a multispectral camera will be embedded / suspended on a drone to capture georeferenced images of fields' crops, for these images to be post-processed for extraction of useful information. The combination of drones and multispectral cameras, enables timely and informed decisions on crop management, given that an appropriate post-processing of the captured photos is conducted. One of the key advantages of such a drone-based toolbox is scalability, as drones can perform flights over large fields quickly in order to collect multispectral data from a variety of angles.

The fundamental operational principles of the DRD toolbox are two-fold. Initially, a pilot with an appropriate training is going to fly the drone over zones of interest and capture georeferenced photos of the zones' states. The georeferenced captured photos will be initially post-processed to match the satellite earth observation data, offered by the EOD toolbox. From this point onwards, the post-processing of the captured photos will compute a variety of vegetation health indices that can be extracted from the captured photos. Details about the implementation of the DRD are presented in the following sections.

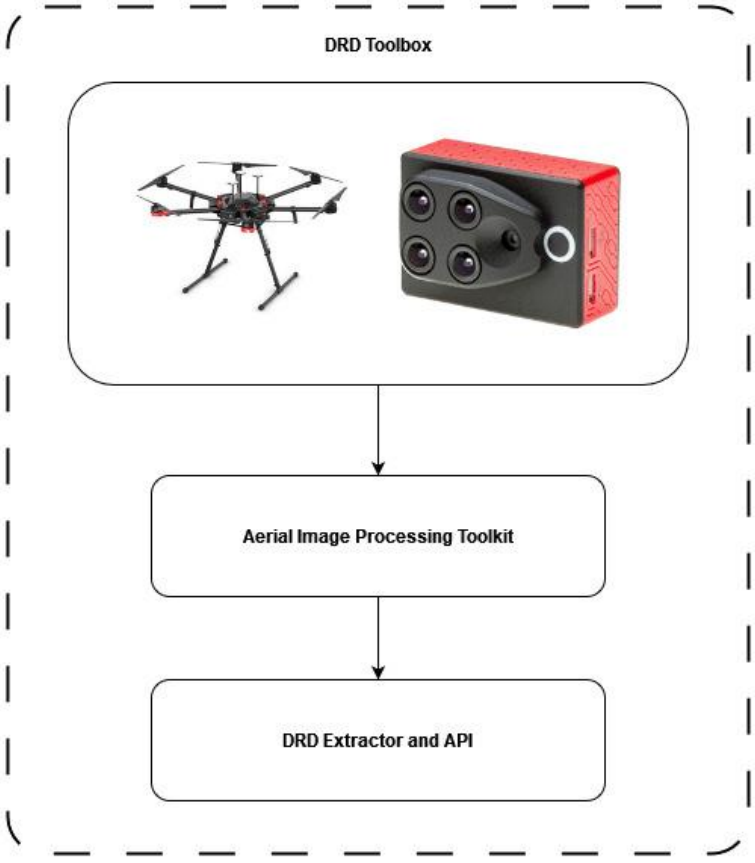


Figure 11: DRD Toolbox component diagram

3.3.2 Development view

3.3.2.1 Component diagram

DRD toolbox is composed of four sub-components, a drone, a multispectral camera, the Aerial Image Processing Toolkit (AIPT) and the DRD Extractor and API. The drone and the multispectral camera are the components responsible for the collection of field images, while the AIPT performs the essential post-processing on the captured data, computing various vegetation indices and other features. Finally, the DRD Extractor and API component serves as an interface to other components of ADV. **Error! Reference source not found.** depicts the internal architecture of the DRD toolbox. More information about the subcomponents can be found in the following sections.

3.3.2.2 Building blocks

3.3.2.2.1 Drone

Naturally, the drone is one of the main building blocks of the DRD toolbox. A drone that is currently available and intended for use within the context of ADV is the DJI Matrice 600 Pro[19]. This drone does not come with an embedded camera, yet its design and high payload capacity allow for the stable suspension of external cameras. This model is equipped with six rotors that provide redundancy and stability, allowing for safer and more reliable flight operations. Most importantly, the drone supports multiple GPS systems for enhanced accuracy. As the project progresses, and the requirements of the DRD toolbox become more apparent, additional drone solutions might be investigated.

Figure 12: DJI Matrice 600 Pro

3.3.2.2.2 Multispectral Camera

The multispectral camera is probably the most essential component of the DRD toolbox. A multispectral camera captures different spectra or images beyond the range visible to the human eye, while multispectral images are widely used in a series of remote sensing applications. A camera that is currently available is the Parrot Sequoia+. This camera captures data in multiple spectral bands, including RGB and Near Infrared (NIR), thus allowing for detailed analysis of vegetation health. Moreover, when combined with the Sunshine Sensor, this camera captures and produces geotagged photos, thus automatically resolving the geo-reference part of the essential post-processing procedures. Once again, though Parrot Sequoia is going to be the starting point of the DRD toolbox, other solutions might be investigated depending on the various use case needs.

Figure 13: Parrot Sequoia+

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

3.3.2.2.3 Aerial Image Processing Toolbox (AIPT)

The Aerial Image Processing Toolbox (AIPT) is the component of DRD that will perform the post-processing of the captured aerial images. AIPT will initially focus on the computation of various vegetation indices. These indices can be extracted, leveraging the spectral bands that are captured by the multispectral camera, such as RGB and Near-Infrared (NIR). An indicative list of vegetation indices (VIs) follows:

- **Normalized Difference Vegetation Index (NDVI):** NDVI is an index that is widely used to assess the health and density of vegetation, with higher values indicating healthier vegetation. It utilizes the Near Infrared (NIR) and red bands. NDVI can be computed as follows:

$$NDVI = \frac{NIR - Red}{NIR + Red}$$

- **Normalized Difference Water Index (NDWI):** NDWI is used to detect the presence of water in vegetation. It is sensitive to changes in water content, helping to identify water bodies while simultaneously monitoring changes in soil moisture. Its computation is based on the NIR and green bands:

$$NDWI = \frac{Green - NIR}{Green + NIR}$$

- **Enhanced Vegetation Index (EVI):** EVI is an improvement of NDVI, designed to minimize the influence of atmosphere and soil background noise. As such, it provides a more accurate representation of vegetation health. The NIR, red and blue bands are utilized for its computation:

$$EVI = 2.5 \times \frac{NIR - Red}{NIR + 6 \times Red - 7.5 \times Blue + 1}$$

- **Green Normalized Difference Vegetation Index (GNDVI)**: GNDVI is an index similar to NDVI, but it utilizes the green band instead of the red. It is useful for assessing vegetation in areas with high soil reflectance, and is computed as follows:

$$GNDVI = \frac{NIR - Green}{NIR + Green}$$

The initial version of the AIPT will focus on the computation of these indices. However, additional post-processing steps might be taken according to cater for other ADV components' needs, possibly implementing a feature extraction procedure if/when essential.

3.3.2.2.4 DRD Extractor and API

The DRD Extractor and API is the component of the DRD Toolbox that will serve as an interface to other components of ADV. The API is going to offer endpoints for the retrieval of computed image features and indices. Moreover, depending on other ADV components' needs, this component will offer the capability of batch extraction of computed features and indices, as well as potentially the publishing of those features to the central Kafka message bus.

3.3.3 Process view

3.3.3.1 Sequence diagram

A sequence diagram of the DRD toolbox is depicted in Figure 14. The flow is quite simple, and it begins with the capturing of aerial geotagged images with the help of the drone and a multispectral camera. The collected images are fed as input to the Aerial Image Processing Toolkit (AIPT). After the post-processing of the captured

images is completed, various computed features become available to other ADV components via the DRD Extractor and API.

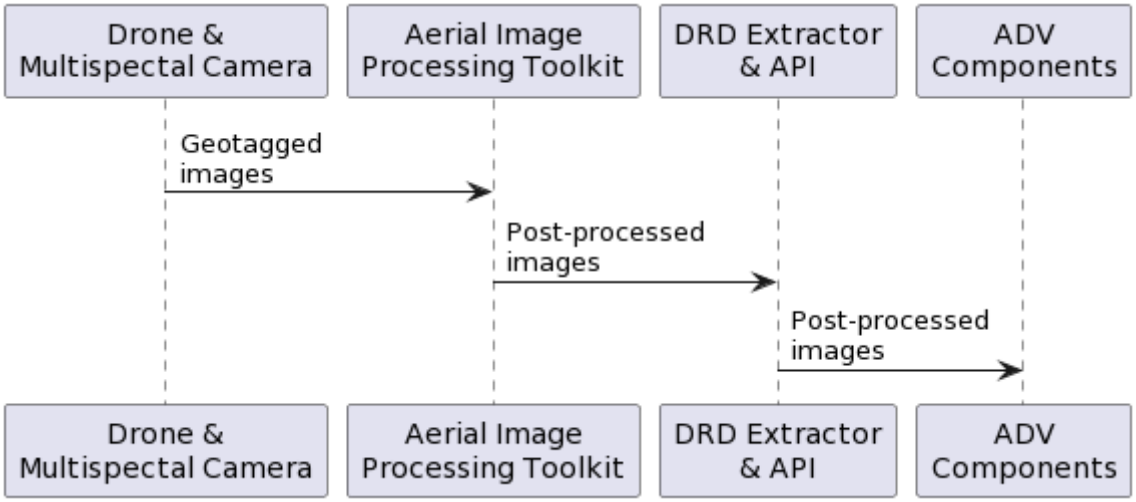


Figure 14: DRD Toolbox sequence diagram

3.3.4 Interfaces

3.3.4.1 Data models used in interfaces

The DRD toolbox will adopt the ADV Data Model, mainly with respect to the communication of image features and indices, to be then processed by other ADV components. Since the toolbox, as well as the ADV data model, is under development and not yet in a stable version, we skip the definition of interfaces for deliverable D4.1.

3.3.4.2 Description of APIs

The DRD Toolbox's API will offer a collection of endpoints for the retrieval of the computed features and indices. As the toolbox is currently under development and an API is not yet available, a detailed description of the offered API will be included in deliverable D4.1.

3.3.5 Technologies and implementation details

This paragraph presents a brief description of the technologies that will be employed for the implementation of the DRD toolbox. To begin with, as already mentioned in the previous paragraphs, the drone that will be used in the context of the toolbox is the DJI Matrice 600 Pro[20], and it will be combined with the Parrot Sequoia+[21] multispectral camera.

The AIPT will be generally implemented in Python, utilizing various computer vision and image processing packages, such as OpenCV[22], Pillow[23], and Numpy[24].

Finally, the DRD Extractor and API will be implemented in Python, and specifically Django and Django REST Framework. Essential packages that will be utilized include Celery, GDAL, and PostGIS.

[20] https://www.dji.com/gr/matrice600-pro
[21] https://www.parrot.com/en/support/documentation/sequoia
[22] https://opencv.org/
[23] https://python-pillow.org/
[24] https://numpy.org/

3.4 Satellite Earth Observation Data Capturing toolbox (EOD)

3.4.1 Description

The Satellite Earth Observation Data Capturing Toolbox (EOD) will utilize the Sentinel Hub platform which is a comprehensive geospatial platform and service provider that specializes in processing, analysing, and delivering satellite imagery and other Earth observation data, including Drone imagery. It offers a wide range of services and functionalities to support various applications in fields such as agriculture, environmental monitoring, disaster management, urban planning, and more.

It enables access to Earth observation (EO) data, primarily from Copernicus satellites (Sentinel 1, Sentinel 2, ...), while also supporting other sources like Landsat, Modis, Planet, and more – a full list of available satellite-based and other raster datasets can be found here at the sentinel documentation webpage[25]. It also enables users to add their own data to the platform, which can be leveraged for the Drone Data Toolbox as well. By utilizing cloud infrastructure and innovative techniques, it efficiently processes and delivers data within seconds. This capability can be seamlessly integrated into any web mapping application, offering a user-friendly and cost-effective solution for utilizing the data within the means of the project.

Sentinel Hub APIs (Application Programming Interfaces) are a set of application programming interfaces (APIs) that enable developers to access, retrieve interact with and utilize satellite imagery and other geospatial data provided by Sentinel Hub. The main APIs offered by Sentinel Hub are:

- Sentinel Hub Process API: Enables users to perform advanced processing tasks on satellite data. It supports the creation of custom image processing chains using a wide range of algorithms, including filtering, band math, index computation, etc.
- Sentinel Hub Batch API: The Batch API allows users to execute large-scale processing tasks on a collection of satellite images. It provides a mechanism for efficient batch processing, enabling tasks such as generating time-lapse animations, running time-series analysis, and performing machine learning workflows on large numbers of images.
- Sentinel Hub Statistical API: The Statistical API allows users to extract statistical information from satellite imagery without the need for downloading the imagery. It provides functionalities like calculating pixel values within specific regions of interest and generating statistical summaries such as mean, median, standard deviation, and percentiles.
- Sentinel Hub BYOC API: The BYOC (Bring Your Own Collection) API allows users to leverage their own imagery data within the Sentinel Hub platform. BYOC enables the integration and analysis of custom data collections alongside the vast repository of Sentinel satellite data. By leveraging the BYOC API, users can tap into the advanced processing capabilities of the Sentinel Hub platform. Users can upload, manage, and access their own imagery datasets, such as high-resolution satellite images or aerial photographs, and seamlessly combine them with the existing Sentinel satellite data for analysis and visualization. The entirety of the geospatial raster dataset generated during the project will be accessed through the Sentinel Hub BYOC API.
- Sentinel Hub Catalog API: Sentinel Hub Catalog API (or shortly "Catalog") is an API implementing the Spatio Temporal Assets Catalog (STAC) Specification[26], describing geospatial information about different data used with Sentinel Hub. The Catalog API enables users to query and access information about available satellite imagery products and metadata

[25] https://docs.sentinel-hub.com/api/latest/data/
[26] https://stacspec.org/en

- Open Geospatial Consortium (OGC) services: Sentinel Hub also supports WMS/WMTS/WCS OGC services.

3.4.2 Development view

3.4.2.1 Component diagram

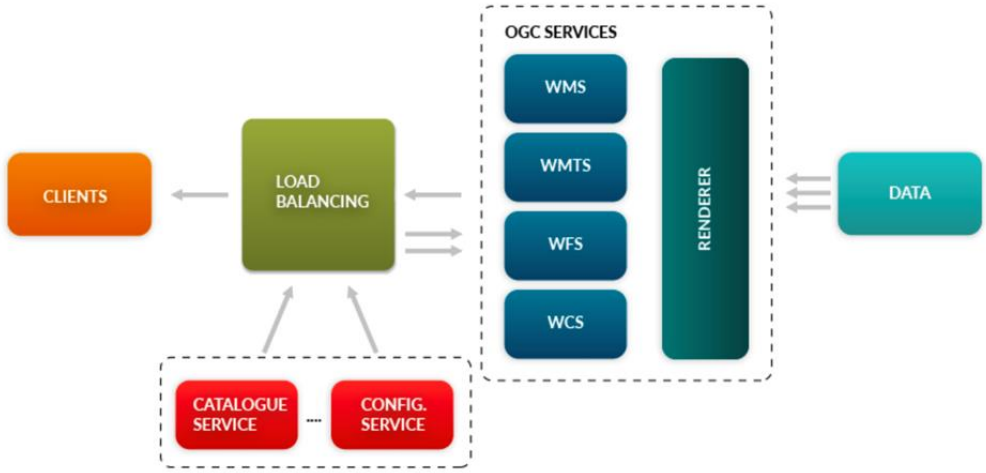


Figure 15: EOD - Component diagram

3.4.2.2 Building blocks

The service's main components are the Catalogue service, Data processing and Rendering service, Custom Scripting engine, and then the Support, POST API and WMS/WMTS/WCS interfaces which provide access to the data and various data manipulation options. In more detail, the main components are:

- Catalogue service. Contains relevant meta-data about all available scenes, allowing efficient (10ms response time) identification of scenes available in the area of interest on the basis of user-defined parameters (time range, cloud coverage, sensor type, etc.).
- Data processing and rendering service. Takes the information of the processing configuration (e.g. combination of bands or more selection of a more complex algorithm) as an input and processes data in real-time, queries the Catalogue service for a chosen AOI, downloads the necessary data from the on-line storage and decompresses it.
- Custom scripting engine. Allows the user to input an algorithm defining pixel-based data processing. A series of open-source custom scripts libraries are available, making it easier for users to start utilizing the library's functionality.
- Data Support. Support is provided for Sentinel-1 GRD, Sentinel-2 (L1C and L2A), Sentinel-3 OLCI, Sentinel-5P, Landsat-5, -7, -8, Envisat MERIS and MODIS data. All products are supported both from the technical aspect of data manipulation (data processing, meta-data), as well as the aspect of implementation (operational deployment).
- POST API and WMS/WMTS/WCS interfaces. The Statistical API provides faster access to higher-level information (e.g. max/min/mean values of a specific index over time).

3.4.3 Process view

In Figure 16 below we show a graphic displaying the role of Sentinel Hub in providing access to various kinds of imagery, such as Open Imagery (i.e. Sentinel-2, Landsat), Commercial Imagery (imagery from commercial satellite data providers), other raster data such as CORINE land cover layers, digital elevation models and similar.

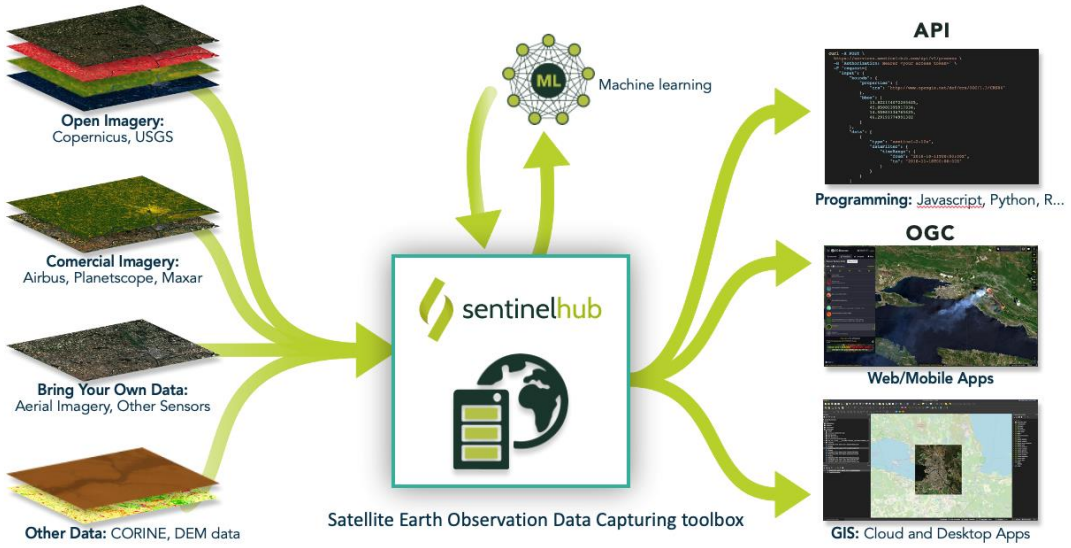


Figure 16: Sentinel Hub diagram. Sentinel Hub allows access to various kinds of raster imagery data (i.e. Copernicus Open Data, custom imagery (Drone), raster model results) via a set of custom OGC compliant API endpoints.

Sentinel Hub also allows ingestion and access to custom raster imagery, such as results of various geospatial models, crop type maps and similar. The access to this imagery is either through Sentinel Hub's RESTful APIs[27] or through it is OGC API, which makes it easy to integrate into various other platforms.

3.4.3.1 Sequence diagram

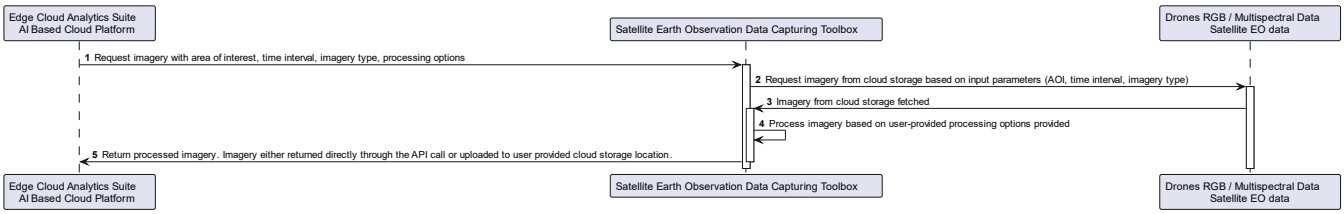


Figure 17: EOD - Sequence diagram

When the EOD Toolbox (which also supports and interacts with the DRD toolbox) gets a request for imagery from other components of the platform (Edge Cloud Analytics Suite, AI Based Cloud Platform) it then processes the request, based on the request parameters, it fetches the appropriate imagery from the cloud storage and processes it appropriately based on the user-provided processing options. The result is then sent back to the requesting component in the format that was requested.

3.4.4 Interfaces

3.4.4.1 Data models used in interfaces

Since Sentinel Hub provides access to petabytes of imagery which can not be JSON serializable, we are providing STAC compliant metadata of the imagery relevant for the other components. STAC is a standardized way to expose collections of spatial temporal data.

[27] https://www.sentinel-hub.com/develop/api/

Sentinel Hub Catalog Collection Info[28]	Provides full list of metadata for the Feature Collection. This information describes the data that is available in a SH collection. It is fully STAC[29] compliant.	
Property	**Type**	**Description**
stac_version	String	The STAC version of the catalog.
Type	String	
id	String	Identifier of the collection used.
title	String	Human readable title of the collection.
keywords	Array of strings	List of keywords describing the collection
license	String	License of the data as a SPDX[30] License identifier.
Extent	Object (CatalogExtent)	The extent of the features in the collection. In the Core only spatial and temporal extents are specified. Extensions may add additional members to represent other extents, for example, thermal or pressure ranges. The first item in the array describes the overall extent of the data. All subsequent items describe more precise extents, e.g., to identify clusters of data. Clients only interested in the overall extent will only need to access the first item in each array.
providers	Array of objects	A list of providers, which may include all organizations capturing or processing the data or the hosting provider. Providers should be listed in chronological order with the most recent provider being the last element of the list.
Summaries	Dictionary	Summaries are either a unique set of all available values or statistics. Statistics by default only specify the range (minimum and maximum values) but can optionally be accompanied by additional statistical values. The range can specify the potential range of values.
Links	Object	Links (with some metadata) to the APIs where the associated item can be retrieved.

[28] https://docs.sentinel-hub.com/api/latest/reference/#tag/catalog_collections/operation/getCatalogCollection
[29] https://stacspec.org/en
[30] https://spdx.org/licenses/

Sentinel Hub Catalog Tile Info[31]	Provides information about a specific tile from the collection.	
Property	**Type**	**Description**
stac_version	String	The STAC version of the catalog.
Type	String	
id	String	Identifier of the tile.
bbox	Array of numbers	The bounding box is provided as four or six numbers, depending on whether the coordinate reference system includes a vertical axis (elevation or depth): • Lower left corner, coordinate axis 1 • Lower left corner, coordinate axis 2 • Lower left corner, coordinate axis 3 (optional) • Upper right corner, coordinate axis 1 • Upper right corner, coordinate axis 2 • Upper right corner, coordinate axis 3 (optional) The coordinate reference system of the values is WGS84 longitude/latitude (http://www.opengis.net/def/crs/OGC/1.3/CRS84).
geometry	Geometry	Geometry of the tile provided as a GeoJSON object.
type	String	The GeoJSON type.
properties	Object (CatalogProperties)	Provides the core STAC metadata fields (i.e. datetime) plus extensions
Summaries	Dictionary	Summaries are either a unique set of all available values or statistics. Statistics by default only specify the range (minimum and maximum values), but can optionally be accompanied by additional statistical values. The range can specify the potential range of values.
Assets	Object (Catalog Assets)	Provides additional information about the assets.

3.4.4.2 Description of APIs

The EOD API description can be found in Annex I while the BYOC and Catalog APIs are described in detail in the public documentation of the service:

- BYOC: https://docs.sentinel-hub.com/api/latest/api/byoc/
- Catalogue API: https://docs.sentinel-hub.com/api/latest/api/catalog/

[31] https://docs.sentinel-hub.com/api/latest/reference/#tag/catalog_item_search/operation/getCatalogItemSearch

3.4.5 Technologies and implementation details

Sentinel Hub is a proprietary platform for which the implementation details are not shared publicly. We do however offer a suite of open-source Python frameworks to make processing of earth observations data obtained from Sentinel Hub easier. These are:

- eo-learn[32] is a collection of open-source Python packages developed to seamlessly access and process spatio-temporal image sequences acquired by any satellite fleet in a timely and automatic manner. It was developed within the PerceptiveSentinel H2020 project. It is easy to use, modular, and encourages collaboration – sharing and reusing of specific tasks in a typical EO-value-extraction workflow, such as cloud masking, image co-registration, feature extraction, classification, etc. The eo-learn library heavily utilizes the following technologies: Python (with libraries for scientific computing), GDAL, PROJ. Also heavily used is the standard python stack of geospatial tools: geopandas[33], rasterio[34].
- eo-grow[35]: Although eo-learn has the functionality to scale up, it is not enough. Issue that we have repeatedly run into was the reproducibility and traceability of the experiments. On top of that, we wanted the capability of coordinating several machines to do the work over large areas. The tagline of eo-grow library is "Earth observation framework for scaled-up processing in Python". Working with EO data is facilitated by the eo-learn package, while the eo-grow package takes care of running the solutions at a large scale. eo-grow library has been developed within the scope of the Global Earth Monitor (GEM) Horizon project. The technology utilized by eo-grow for scaling is Ray — Ray is an open-source unified compute framework that makes it easy to scale AI and Python workloads — from reinforcement learning to deep learning tuning, and model serving.

[32] https://github.com/sentinel-hub/eo-learn
[33] https://geopandas.org/en/stable/
[34] https://rasterio.readthedocs.io/en/stable/
[35] https://github.com/sentinel-hub/eo-grow

4 Edge Cloud Analytics Suite

The Edge Cloud Analytics Suite aims at training and serving Decision Support Systems (DSS), particularly focusing on the realm of Artificial Intelligence (AI), and more specifically Deep Learning (DL) models, tailored for applications within the agri-food supply chain. To generate semi-trained models leveraging in-situ, regional, and global data across diverse entities in an efficient, scalable, and secure way, AgriDataValue advocates for the adoption of a Federated Deep Machine Learning (FDML) strategy combined with Privacy-preserving Machine Learning (ML) algorithms. In this way, AgriDataValue will enable distributed training of models at the edge without necessitating data sharing among users, while avoiding the leakage of private information throughout the process. It will also enable the combination of localized models into a comprehensive global model that gathers knowledge from diverse data sources. Finally, this framework will facilitate the usage of these global models at the edge while maintaining minimal processing requirements.

Furthermore, in a concerted effort to enhance the platform's user experience and instill confidence in its offerings, AgriDataValue introduces Explainable AI (XAI) predictive modeling. This augmentation aims to furnish transparency to the federated-trained algorithms, thereby empowering users with insights into model decisions incrementing the trust in the platform, its models, and subsequent recommendations.

In this way, the functionality of the Edge Cloud Analytics Suite is divided into two different components reflected in the initial architecture of the platform: FDML and XAI, described in more detail in Sections 4.1 and 4.2 respectively.

4.1 Federated Machine Learning (FDML)

4.1.1 Description

The FDML component offers a framework designed to train and serve AI and Deep Learning (DL) models, with data from diverse sources and entities.

Federated Learning (FL) [4] is a distributed approach that enables training models across several decentralized edge devices that possess private local data, avoiding exchanging any private information. In this way, instead of centralizing data on a single server, FL allows model training to occur on individual devices housing the respective data. The conventional FL strategy involves two distinct types of agents (usually called FL agents): the clients and the server. Clients refer to devices with access to private local data and with sufficient computational capabilities to conduct ML model training. After training this model, usually referred to as local models, the clients share their model parameters or weights with the server. On the other hand, the server is responsible for orchestrating the FL process. It generally initializes the model and sends it to the clients. Once the local models are received, the server aggregates these models to create a model with knowledge earned from all the data sources. This model is generally referred to as the global model.

Specifically, AgriDataValue proposes a Hierarchical Federated Learning (HFL) [5] strategy to enhance the learning process. HFL extends the traditional FL approach by introducing a hierarchical structure among participating devices. As can be seen in Figure 18 and Figure 19, rather than a flat structure where all clients connect directly to a central server, HFL organizes devices into layers or levels.

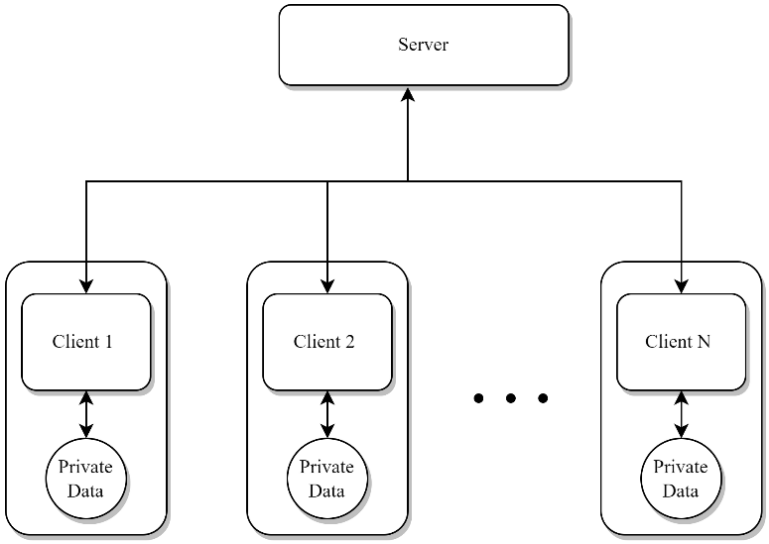


Figure 18: Conventional FL example architecture

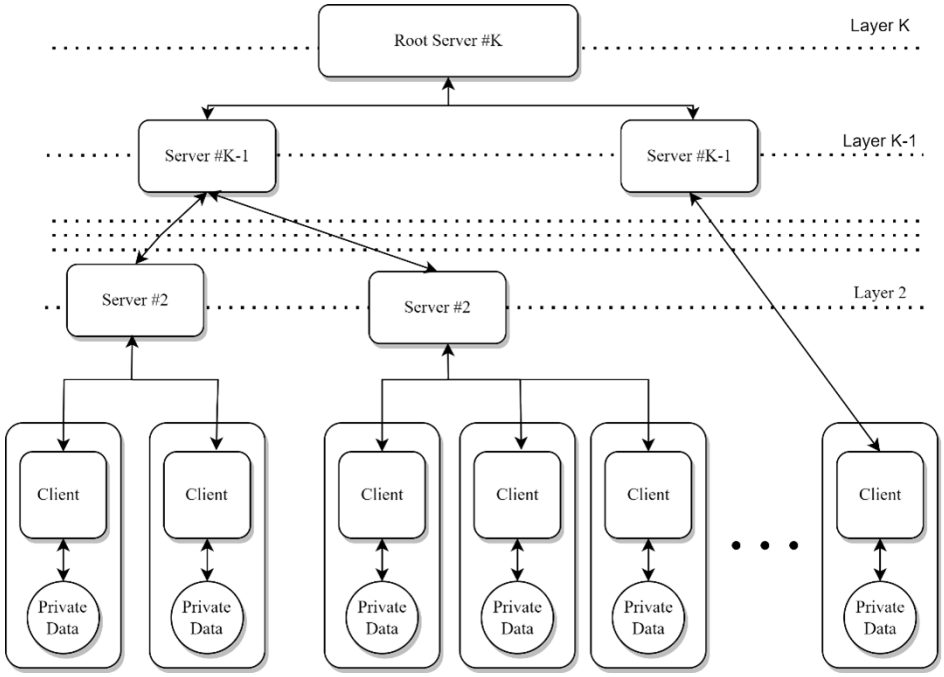


Figure 19: FDML -Generic HFL topology

In this way, servers in an HFL strategy orchestrate a FL process wherein the clients are devices within a lower layer of the hierarchy, and the aggregated model is sent to the associated server in a higher layer. It is important to remark that the top-tier server in the hierarchy, typically known as the root server, manages the process of global model generation, often referred to as cross-silo FL. In contrast, intermediate servers, positioned in the middle layers, orchestrate intra-silo FL.

Establishing this hierarchy optimizes training efficiency in terms of computational resources and bandwidth by minimizing data transfer to higher levels including the root server. Additionally, it enhances scalability by distributing the workload across multiple levels, enabling the FL procedure to handle a larger number of clients. Ultimately, incorporating this hierarchy can significantly reduce the training time required for the generation of the global model.

AgriDataValue initially proposes a three-level hierarchy depicted in Figure 3. This network topology comprises: first, a client layer responsible for local model training deployed at the edge; second, a layer of intermediate servers that will be strategically positioned to communicate with nearby clients, and that aggregates local models to generate what we will refer as regional models; finally, the central server within the Decentralised Knowledge Management (DKM) component (refer to Section 6.1 for more details) is in the cloud. This central server aggregates regional models into a global model. The initial architecture shown in the figure serves as an example with four clients, two intermediate servers, and a central server. However, it is worth noting that this architecture may vary based on the available node set and the data source's location.

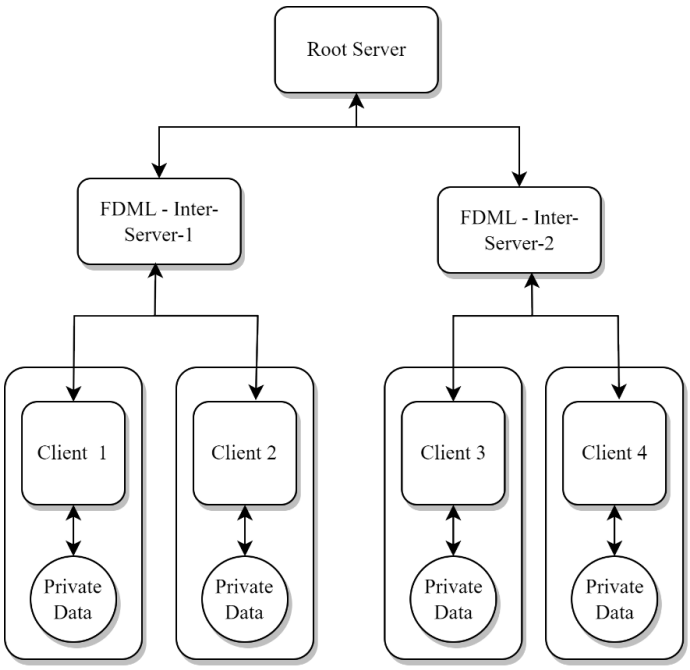


Figure 20: AgriDataValue FDML initial hierarchy

Subsequent sections will provide a more detailed description of the sub-components and functionalities of the FDML component, including a detailed specification of the technologies employed and their development.

4.1.2 Development view

4.1.2.1 Component diagram

Figure 21 shows the component diagram for the FDML, representing its several sub-components and the several functional blocks. Even if they are not considered components, these functional blocks are crucial to correctly understand the complete scope of the component. For the sake of simplicity, only one FDML client and a single intermediate server out of those that conform the whole FDML architecture (schematized in Figure 20) are depicted. It is important to remark that the internal structure of these sub-components remains consistent for all the FDML clients and intermediate servers.

To facilitate understanding, sub-components within the FDML are represented with solid lines, functional blocks with dashed lines, and components outside the FDML are grayed out. Additionally, arrows denote direct communication or interaction between components, with variations in their design based on the process or information flow being carried out.

AgriDataValue platform initially considers two distinct flows:

1. **Training or fulfillment flow:** This process encompasses training and storing models within the platform. It starts with data acquisition via the Decentralized Data Capture Management toolbox

(section 6.1) and concludes with model and metadata storage in the SECURESTORE (Section 5.1). The FDML's contribution to this process is represented using solid lines.

2. **Inference or delivery flow:** In this process, the platform returns advice or prediction to the end user given a new set of data. This process starts with data acquisition via the Decentralized Data Capture Management toolbox and culminates in delivering the prediction along with its explainability by the XAI component. This flow is depicted using dashed lines.

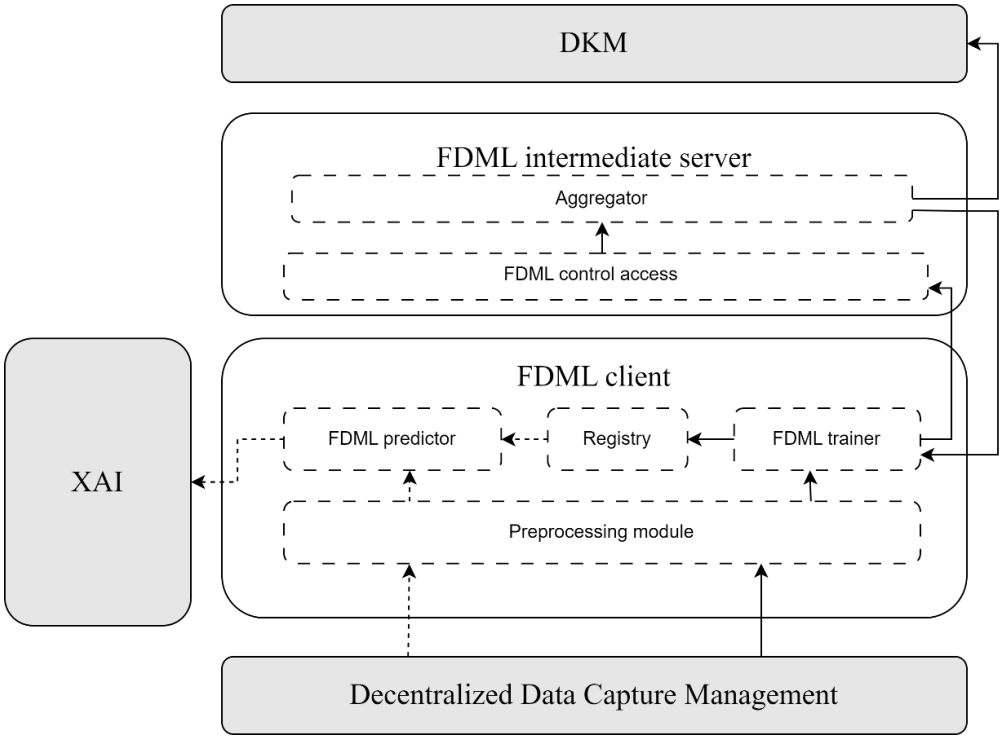


Figure 21: FDML component diagram

It is worth noting that this diagram mainly focuses on the internal connections within FDML and the direct connections between FDML and other components. The next sections further analyze the components and functional blocks represented here, and the communication protocol employed.

4.1.2.2 Building blocks

As shown in Figure 21, the FDML component is divided into two sub-components: the FDML client and the FDML intermediate server, each presenting different functional blocks.

1. **FDML clients:** The functionality of this sub-component depends on the process being executed. During the training process, the FDML clients take as input the data from the Decentralized Data Capture Management toolbox in the specified AgriDataValue format and conduct local AI model training following the federated scheme described in section 4.1.1. In the inference process, aiming at maintaining the decentralized structure of AgriDataValue, the FDML clients are responsible for making predictions. Therefore, given new incoming data, the FDML client sends the prediction made with the previously generated global model to the XAI component. These FDML clients are deployed on edge devices, granting access to data from the pilots.

   To reach these objectives, the functionality of this component is divided into several blocks:

   - *Preprocessing module:* this module adapts input data to the selected AI model. Considering the vast heterogeneity of data supported by AgriDataValue, this preprocessing module must handle

different data types, including time series, images, or tabular data. Additionally, it computes several statistics such as the mean or the standard deviation from the training data.

- *FDML Trainer:* this block conducts DL model training with the preprocessed data and shares the model weights with the associated intermediate server. Even though the model parameters do not directly contain training data information, it has been demonstrated that attacks can extract some insights using generative-based techniques. Hence, upcoming platform versions will explore privacy-preserving techniques, particularly based on differential privacy, to avoid any data leakage from the weights. Examples include strategies like PATE (Private Aggregation of Teacher Ensembles) [6] , secure sum, or differential privacy-based model optimization algorithms such as DP-SGD [7] [8].
- *Registry:* This block records trained models and preprocessing conducted.
- *FDML predictor:* This functional block executes the inference process associated with the selected model and returns the prediction obtained to the XAI component. In this way, explainability is included in both the model and the prediction.

2. **FDML intermediate server:** This sub-component corresponds to the intermediate server in the HFL strategy described before. Unlike FDML clients, the functionality of these components is confined to the training process, orchestrating the intra-silo FL with the associated nearby geospatial FDML clients. Once the intra-silo FL process is completed, the FDML intermediate server sends the regional model weights to the root server located in the DKM for generating the global model. Finally, when the FDML intermediate server receives the global model, it forwards the weights to its associated clients.

   To fulfil this functionality, the FDML intermediate server comprises two functional blocks:

   - *FDML control access:* This block verifies that the origin of the weights corresponds to a device associated with the server. In this way, the platform proposed is robust to potential attacks from unregistered malicious devices.

Aggregator: this block aggregates all benign local models. It is crucial to note that the initially considered aggregation technique is Federated Averaging (FedAvg) [9], as it is the most used in the state-of-the-art. However, throughout the project, we will explore the incorporation of other aggregation techniques.

4.1.3 Process view

4.1.3.1 Sequence diagram

Figure 23 and Figure 24 depict the sequence diagram illustrating the functionalities and intercommunication among the sub-components within the FDML component during both the training and inference processes, respectively. This section describes these processes in more detail, while subsequent sections describe the interfaces and data models involved. It is worth noting that all communications operate via the HTTP protocol, exposing a REST API.

Regarding the training or fulfilment flow, the iterative process of FL can be segmented into the following steps:

1. **Model initialization:** The DKM (Section 6.1) initiates the learning process of the DL model. Once the model is defined and initialized, the DKM forwards it throughout the defined network topology for the learning process.
2. **Intra-silo FL:** in this iterative process, intermediate servers and FDML clients engage in the regional model generation process, which consists of the following sub-steps.
   a. **Local private training:** FDML clients access data from the Data Capture Management component, preferably through an API. However, alternative data access methods will be

explored during the project. Then, clients train their local models with their respective data, assess performance on a validation subset, and save/register the best-performing model.

b. **Regional model generation:** clients forward local models to their respective intermediate FDML servers. These intermediate servers validate the origin of the received models and perform FedAvg aggregation to generate regional models.
c. **Regional model evaluation:** Regional models are sent to FL clients for storage and evaluation against their validation data.

3. **Global model generation:** Upon completion of the intra-silo FL, FDML intermediate servers forward regional models to the DKM. Finally, following the process described in Section 6.1, the DKM aggregates these models into the global model.
4. **Global model evaluation:** Finally, the global model is sent back to clients. This enables clients to make predictions directly on-device.

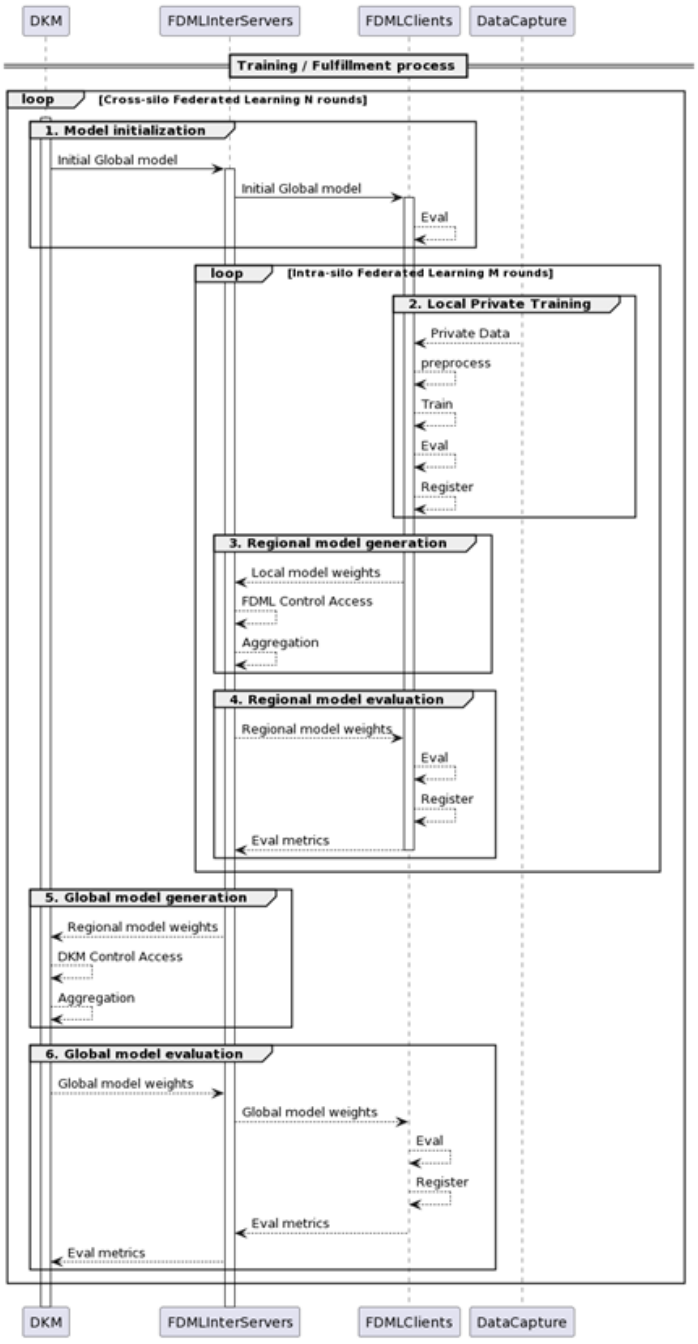


Figure 22: Process view of the FDML for the training or fulfilment process

Regarding the inference or delivery process, AgriDataValue proposes a decentralized platform where predictions or advice returned are processed directly at the edge, by the nodes within the platform. This process is shown in Figure 23 and can be divided into the following steps:

1. **Request for advice:** The end-user, via the AgriDataValue frontend or a POST request, requests advice or a prediction from the platform. The request must contain an input dataset in JSON format that will be used by the platform to make a prediction.
2. **Making a prediction:** The prediction is received by an FDML client responsible for selecting the model for inference from its internal registry, preprocessing the input data (through the preprocessing functional block) and make the prediction (through the FDML predictor functional block).
3. **Adding explainability:** Upon obtaining the prediction, the FDML client sends the obtained prediction along with the ID of the used model to the XAI service. Following the process described in Section 4.2, the XAI service adds necessary explainability to the model and prediction. This enables the end-user to better understand and interpret the prediction.

It is important to emphasize that in this process, neither the DKM nor the intermediate servers of the FDML are involved.

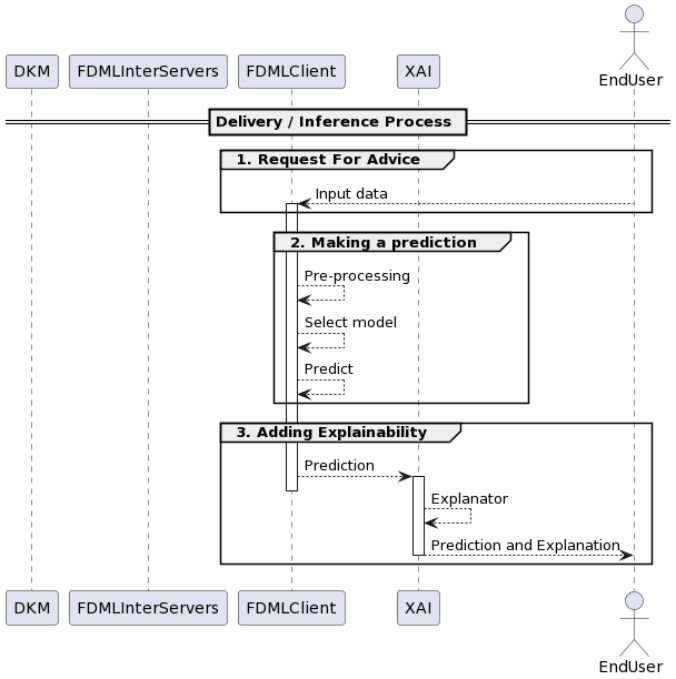


Figure 23: FDML - Process view of the FDML for the inference or delivery process

4.1.4 Interfaces

4.1.4.1 Data models used in interfaces

Table 1: Data models used in the FDML

Name	FDML Model Interface	
Property	**Type**	**Description**
Weights	List of floating-point arrays	Serialized model parameters
N	Int	Number of samples of the training dataset

4.1.4.2 Description of APIs

All the sub-components of the FDML expose a REST API described in the following tables. A wide description of the functionality associated with these interfaces is available in Annex I.

1. **FDML Clients APIs**

Table 2: Description of the interfaces of the FDML clients

Title	Aggregated model interface
URL: *This field holds the relative path to the described API. For simplicity Root path can be cut off from this description and can be placed as a hypertext above the API template*	
/rest/downstream	
Method *This field holds the type of the Method used*	
POST	
URL Params *This field holds the parameters (if any). Separated based on the fields below into required and optional.*	
Data Params *This field holds the body payload of a post request.*	
Required:	
Weights[array of floats]	*Parameters of the local model.*
Optional:	
N[int]	*Number of samples of the local training dataset*
Success response <What should the status code be on success and is there any returned data? This is useful when people need to know what their callbacks should expect>	
200	*Local model correctly trained*
Error response *This field holds the list of all possible error responses. Doing that, helps prevent assumptions of why the endpoint fails and saves a lot of time during the integration process.*	
404	*The FDML client is not running*
400	*Syntax error in the post command*
Sample call *This field holds a possible sample call to the described endpoint in a curl-like format. Please, choose the format wisely so that is clear and easy to read by the interested parties.*	
curl --location '0.0.0.0/rest/downstream' \ --header 'Content-Type: application/json' \ --**data** '{ "weights":[[0.03,0.21,-0.03]] }'	

2. **FDML intermediate servers APIs**

Table 3: Description of the interface/rest/downstream of the FDML intermediate server

Title	Broadcast global model
URL: *This field holds the relative path to the described API. For simplicity Root path can be cut off from this description and can be placed as a hypertext above the API template*	
/rest/downstream	
Method *This field holds the type of the Method used*	
POST	
URL Params *This field holds the parameters (if any). Separated based on the fields below into required and optional.*	
Data Params *This field holds the body payload of a post request.*	
Required:	
Weights[array of floats]	*Parameters of the local model.*
Optional:	
N[int]	*Number of samples of the local training dataset*

Success response <What should the status code be on success and is there any returned data? This is useful when people need to know what their callbacks should expect>	
200	*Global model correctly broadcasted*
Error response *This field holds the list of all possible error responses. Doing that, helps prevent assumptions of why the endpoint fails and saves a lot of time during the integration process.*	
404	*The FDML intermediate server is not running*
400	*Syntax error in the post command*
Sample call *This field holds a possible sample call to the described endpoint in a curl-like format. Please, choose the format wisely so that is clear and easy to read by the interested parties.*	
curl --location '0.0.0.0/rest/downstream' \ --header 'Content-Type: application/json' \ --**data** '{ "weights":[[0.03,0.21,-0.03]] }'	

Table 4: Description of the interface /rest/upstream of the FDML intermediate server

Title	Broadcast of regional models
URL: *This field holds the relative path to the described API. For simplicity Root path can be cut off from this description and can be placed as a hypertext above the API template*	
/rest/upstream	
Method *This field holds the type of the Method used*	
POST	
URL Params *This field holds the parameters (if any). Separated based on the fields below into required and optional.*	
Data Params *This field holds the body payload of a post request.*	
Required:	
Weights[array of floats]	*Parameters of the regional model.*
Optional:	
N[int]	*Total number of samples of the regional training*
Success response <What should the status code be on success and is there any returned data? This is useful when people need to know what their callbacks should expect>	
200	*Regional model correctly broadcasted*
Error response *This field holds the list of all possible error responses. Doing that, helps prevent assumptions of why the endpoint fails and saves a lot of time during the integration process.*	
404	*The FDML intermediate server is not running*
400	*Syntax error in the post command*
Sample call *This field holds a possible sample call to the described endpoint in a curl-like format. Please, choose the format wisely so that is clear and easy to read by the interested parties.*	
curl --location '0.0.0.0/rest/upstream \ --header 'Content-Type: application/json' \ --**data** '{ "weights":[[0.03,0.21,-0.03]] }'	

4.1.5 Technologies and implementation details

Python 3.11 has been employed for the development and Poetry[36] for the installation and management of the different packages used. Several Python packages, such as Pandas (version 2.0.3) and NumPy (version 1.23.6), have been utilized for data preprocessing. However, it is essential to highlight the usage of TensorFlow as a Deep Learning framework and Fleviden as the Federated Learning framework to implement the hierarchical federated learning.

TensorFlow [10] is an open-source ML framework developed by Google, designed to facilitate the creation, training, and deployment of ML solutions. This framework enables users to construct and train various ML algorithms, including neural networks and DL models, while offering flexibility in deploying these models across different hardware platforms such as CPUs, GPUs, and TPUs.

On the other hand, Fleviden is a fully extensible distributed learning framework developed internally by ATOS's Research and Development department. A more extensive description of the framework and more technical details on the implementation of the FDML components are available in Annex A.

It is important to remark that all the components have been containerized using Docker[37], and their deployment has been performed using Docker compose. However, following AgriDataValue guidelines, the translation of the Docker composes into Kubernetes[38]manifests is underway for handling the deployment on AgriDataValue premises. The developments related to this component have been included in the first technical demonstration of the project, available on the project's GitLab repository[39].

4.2 Human Explainable Conceptual Framework (XAI)

The field of ***eXplainable Artificial Intelligence*** **(XAI)** refers to artificial intelligence (AI) systems that can be easily understood and interpreted by humans, especially non-expert end-users. The goal of XAI is to bridge the gap between the complexity of AI algorithms and human understanding, making AI systems more transparent, accountable, and trustworthy.

In this section the solution provided by AgriDataValue as outcome of the project's work is presented. In the following subsections a description of the solution, its components, its main operational processes, and interfaces is provided.

4.2.1 Description

As part of the AgriDataValue project, which aims to drive digital transformation in agriculture at the European level, a specialized module is being designed and created to increase the trust and confidence of end-users in this technological ecosystem that will make massive use of AI.

The **Human Explainable Conceptual Framework**, i.e. the AgriDataValue solution to explain the decision of AI models, is designed and being developed with two main goals:

1) to make the process of AI algorithms interpretable and explainable

[36] https://python-poetry.org/
[37] https://www.docker.com/
[38] https://kubernetes.io/
[39] https://git.agridatavalue.eu/agridatavalue/demos/demo-fulfillment-flow

2) to ensure that end-users, be them individual farmers or cooperatives, trust the AI decision

The framework features three main characteristics (i) **user–centric design**: interpretability methods of AI models focused on end-users, specifically farmers and farming cooperatives, (ii) **Performance Indicators**: qualitative and quantitative metrics to assess the level of interpretation of an XAI explainer (the interpretation must be as complete as possible, from the data down to the algorithm itself) and (iii) **Likert scale** as measure of end-user satisfaction with the provided explanation.

End-users will benefit of the XAI framework by better understanding the mechanisms behind AI predictions including e.g. crop yield forecasts, disease risks, optimal irrigation schedules etc. As represented in Figure 24, this virtuous circle tends to increase farmers' trust in the adoption of advanced AI technologies thus contributing to the smart-farming objectives towards long-term sustainability.

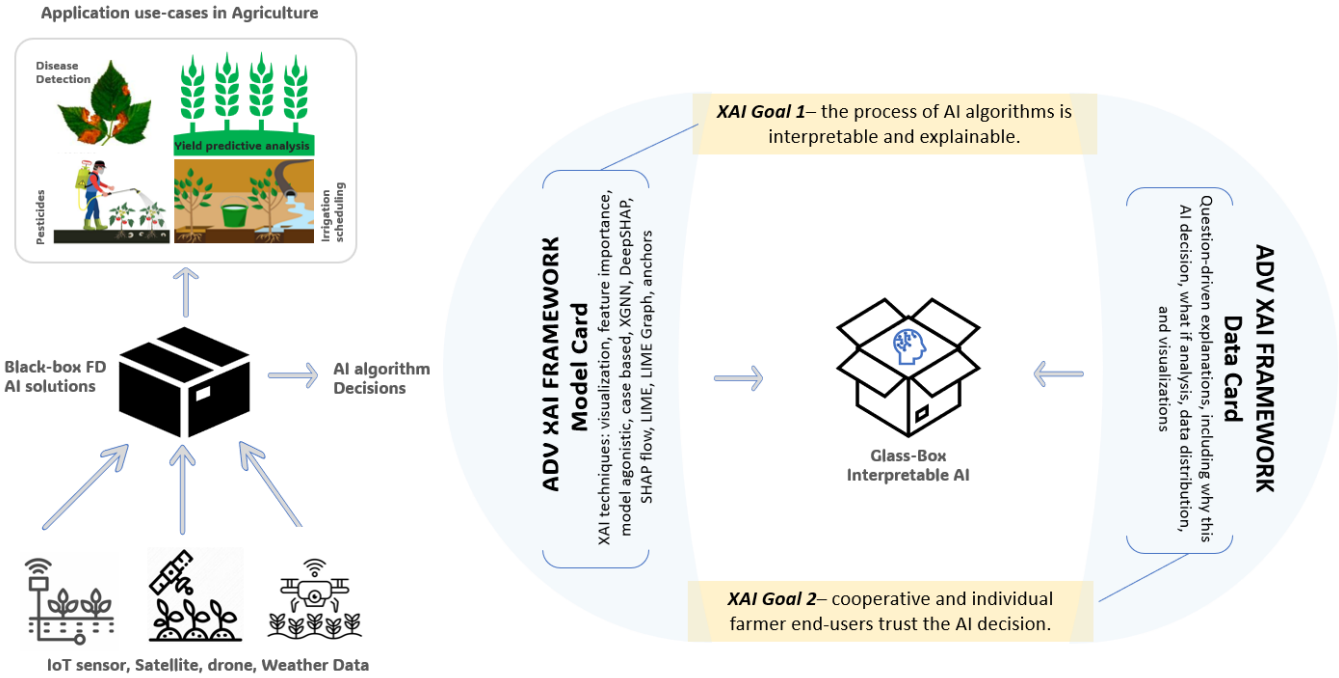


Figure 24: XAI - Framework envisioned to develop AI systems that are transparent and explainable so that they can be trusted by non-expert end-users

4.2.2 Development view

4.2.2.1 Component diagram

The XAI component includes the logical flow diagram that displays the logical architecture of the proposed XAI framework solution at high level, outlining the structure of the internal sub-components and how they interact within two main operational processes:

a) **Fulfillment process**: The production of the information content needed to enable the explanations' delivery.

b) **Delivery process**: the delivery of explanation content to the end user.

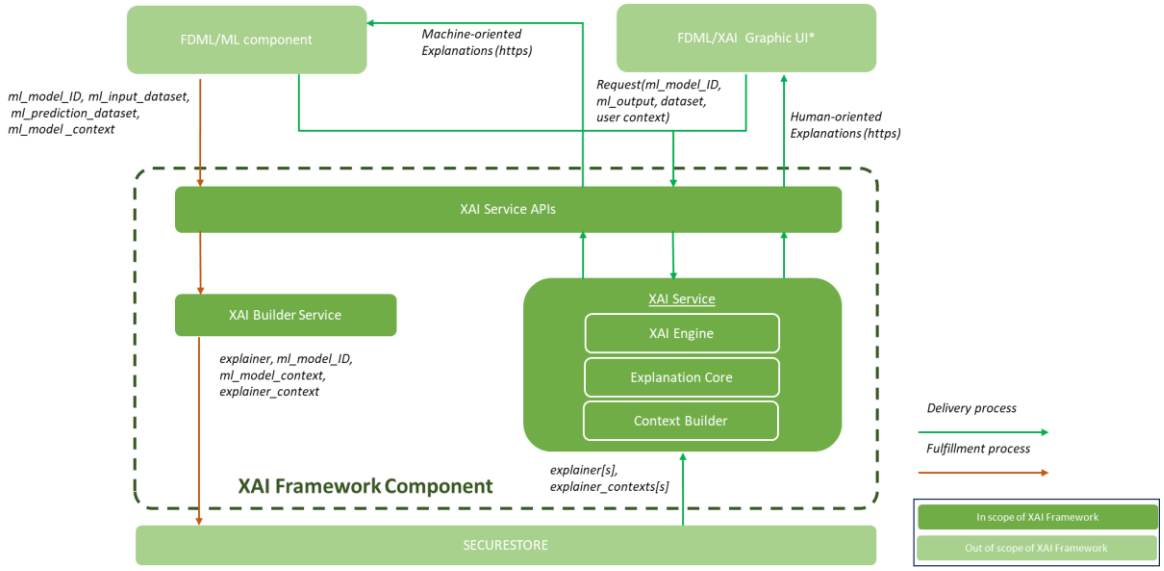


Figure 25: XAI - Framework logical components interacting to assure XAI functionality.

Both processes are activated by calls to a **Service API** layer that exposes the services to interfacing components

Fulfilment process:

- Each time a new ML model is trained, the FDML [40] component sends the information needed to create a new **explainer**, specifically:
  - *ml_model_ID: path to model file*
  - *ml_input_dataset: features representation*
  - *ml_prediction_dataset: predicted target representation*
  - *ml_model_context: path to model metadata*
- This information is processed by the **XAI Builder Service** to produce the corresponding explainer (trained XAI model). At the same time an explainer context is produced, summarizing the explainer characteristics.
- The explainer is then stored in the SECURESTORE associated to the ml_model ID.

Delivery process:

- Requests to the **XAI service** may come from both human users through UI (e.g. end-users, developers) and from programmatic access. The request shall contain reference to the used ML model (ml_model_ID) and the output, data sample and user context.
- The XAI Service retrieves information about the ml_model context and the matching explainer from the SECURESTORE component.
- Based on the explainer an **explanation** is built and returned.

Assumption:

The underlying assumption of this solution is that the XAI framework component and the FDML Component maintain a shared taxonomy and inventory of ML Models. The ML model is produced in the FDML context, but

[40] Along this section 4.2 the inner composition and behaviour of the FDML and the SECURSTORE components is not detailed. For detailed specifications of these components and their interfaces please refer to sections **Error! Reference source not found.** and **Error! Reference source not found.**, respectively.

needs to be known and accessible to the XAI framework through the SECURESTORE layer (e.g. in H5 and PKL format, as applicable)

4.2.2.2 Building blocks

As introduced in section 4.2.1 the deployment of the XAI Framework involves three internal building block sub-components. The **XAI Service API**, which provides endpoint exposure to the DKM component in the delivery process and external end-user actors in the delivery process, **XAI builder**, which processes input information access from SECURESTORE to map the AI model to the corresponding XAI explainer and keeps it back in SECURESTORE under the explainer bucket, the **XAI service**, which retrieves from the SECURESTORE the explainer and the associated metadata for the end-user who initiated the request through the XAI Service API.

4.2.3 Process view

4.2.3.1 Sequence diagram

The sequence diagram at Figure 26 describe the way the XAI component is integrated in the end-to-end process of delivering predictions and relevant explanations to an end user.

The XAI component is activated by the FDML component with the step (8). Within the same step it receives the FDML outcome, i.e. the AI model and its contexts and predictions, needed to explain the decision of the algorithm. At step #9 XAI explanations contextualized to end-users are produced; they can include feature-based explanations, example-based explanations, rule-based explanations, and contextual information. The explanations are then returned to end-user apps/dashboards (step #11).

Preliminarily to this process an internal fulfilment process has took place where the AI models are mapped to their corresponding explainers and stored in SECURESTORE along with their contexts. This process in internal to the application and is visible at Figure 25.

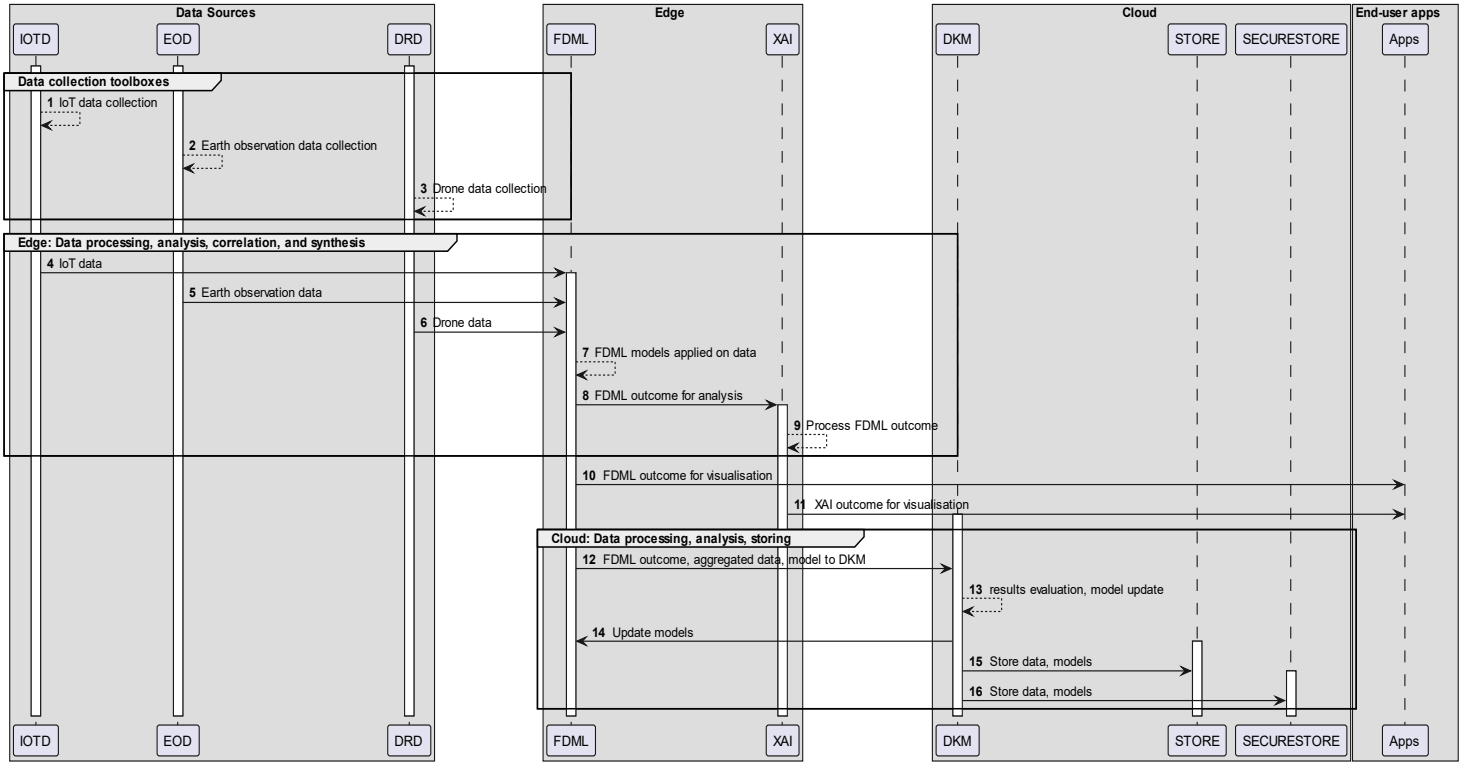


Figure 26: XAI - Sequence diagram integrating XAI and FDML

4.2.4 Interfaces

4.2.4.1 Data models used in interfaces

Name	*XAI Data model*	
Property	**Type**	**Description**
ModelType	Object	Configuration model type
ModelReport	Object	Information about model
DatasetReport	Object	Information about training datasets
OutputServing	Object	Output Serving object
ExplainationServing	Object	Explanation information about local event
UserSatisfaction	Object	XAI Satisfaction object

4.2.4.2 Description of APIs

(REST)

Title	*XAI_Fulfilment API*
URL: /xai/mapper	
Each time a new ML model is trained, the DKM component sends a post request to this endpoint in order to generate a corresponding explainer which will be stored in minio explainers bucket	
Method *This field holds the type of the Method used*	
POST	
Data Params *This field holds the body payload of a post request.*	
Required:	
model=[str]	*Path to the model file in minio, the file should be in "model.pkl"/ "model.h5"*
Required:	
metadata=[str]	*Path to the model metadata file in minio, the file should be in "metadata.json" describing the context of AI model (ml_model_context)*
Required:	
ml_input_dataset=[str]	*The input data used to train the AI model- which will be used by the XAI explainer the explain how the AI model reached at this decision*
Required:	
ml_prediction_dataset: [str]	*The predicted target values - which will be used by the XAI explainer the explain how the AI model reached at this decision*
URL Params *This field holds the parameters (if any). Separated based on the fields below into required and optional.*	
Required:	
Success response <What should the status code be on success and is there any returned data? This is useful when people need to know what their callbacks should expect>	
200 Content: {"status": "success - explainer and metadata stored in minio"}	*A success message indicating the explainer and its associated metadata has been stored in minio explainers bucket*
Error response *This field holds the list of all possible error responses. Doing that, helps prevent assumptions of why the endpoint fails and saves a lot of time during the integration process.*	
404	*response description*

Sample call *This field holds a possible sample call to the described endpoint in a curl-like format. Please, choose the format wisely so that is clear and easy to read by the interested parties.*
curl -X 'POST' \ 'http://localhost:8000/xai/trigger/' \ -H 'accept: application/json' \ -H 'Content-Type: application/json' \ -d '{ "model": "string", "metadata": "string", "ml_input_dataset ": "string", "ml_prediction_dataset": "string" }'
Notes *This field holds any additional helpful info related to this endpoint.*
In the fulfillment process phase of the XAI framework, all the artifacts of the **XAI Builder Services** mapping the AI model to XAI techniques will deposited in SECURE STOR.

Title	*XAI_Delivery API (being done and will updated)*
URL: /xai/explain	
Each time the end user gets a prediction from a trained ML model based on his/her own input data he/she can use this API to get the corresponding explanations on the AI decisions.	
Method *This field holds the type of the Method*	
Get	
Data Params *This field holds the body payload of a post request.*	
Required:	
"model_id": "string"	*Model_id for used for prediction*
Required:	
"dataset": "string"	*Path to the dataset used to get prediction from the model with model_id*
Required:	
"model_output": "dict"	The output result of the prediction for specific user-data a dictionary
URL Params *This field holds the parameters (if any). Separated based on the fields below into required and optional.*	
Required:	
Optional:	
Success response <What should the status code be on success and is there any returned data? This is useful when people need to know what their callbacks should expect>	
200 Content: {"status": "success - explainer and metadata stored in minio"}	*A success message indicating the explainer and its associated metadata has been stored in minio explainers bucket*
Error response *This field holds the list of all possible error responses. Doing that, helps prevent assumptions of why the endpoint fails and saves a lot of time during the integration process.*	
404	*response description*
Sample call *This field holds a possible sample call to the described endpoint in a curl-like format. Please, choose the format wisely so that is clear and easy to read by the interested parties.*	
curl -X 'GET' \ 'http://localhost:8000/xai/explain/' \ -H 'accept: application/json' \ -H 'Content-Type: application/json' \ -d '{ "model_id": "string"	

"dataset": "string" "model_output": "dict" }'
Notes *This field holds any additional helpful info related to this endpoint.*

4.2.5 Technologies and implementation details

The underlying technology for XAI conceptual framework involves various techniques and approaches aimed at providing insights into how AI models make decisions. In AI, Machine learning (ML) algorithms can be categorized as white-box or black-box. White-box models, sometimes called glass box models, provide results that are understandable to experts in the domain. Black-box models, on the other hand, are any AI systems whose inputs and operations aren't visible to the user, or another interested party. The goal of the XAI is to expose black boxes and make them as explainable as a white box. The XAI methods that will be implemented are characterized along two axes:

- **Local** and **global methods**: as the name suggests, these are methods used to explain different aspects or scope of the AI model behaviour. Local explanations explain single model decisions, while global explanations characterize the general behaviour of a model (e.g., a neuron, a layer, an entire network). In some cases, global explanation is derived from local explanations, but this is not necessarily true for all artificial intelligence models.
- **Post-hoc** and **Ante-hoc Methods**: Post-hoc ("after this event") methods are those methods that provide the explanation after the model has been trained with a standard training procedure; examples of such methods are LIME, BETA, LRP. Ante-hoc methods are those that are interpreted immanently in the system, i.e., they are transparent by nature in the sense that these methods introduce a new network architecture that produces an explanation as part of its decision.

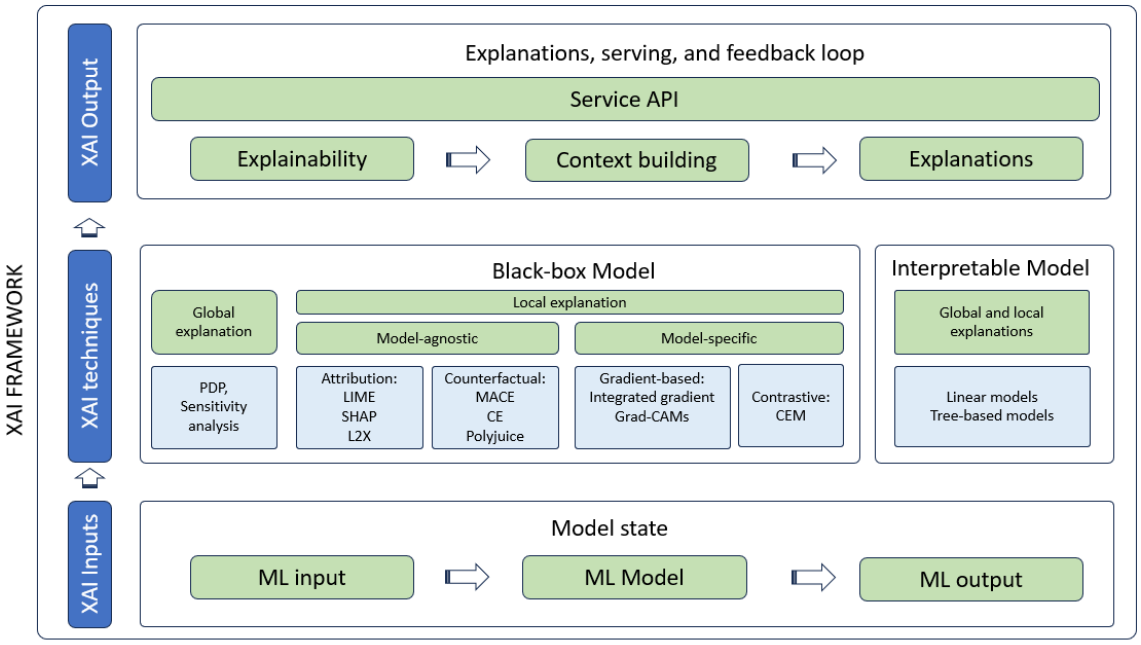


Figure 27: ADV XAI Framework implementation toolsets along with scope of explanation

The most common toolsets and frameworks for developing XAI systems that will be leveraged in XAI framework are described in the figure above and include:

- LIME: Uses interpretable feature space and local approximation with sparse K-LASSO.
- SHAP: Additive method; uses Shapley values (game theory) unifies Deep LIFT, LRP, LIME.
- Anchors: Model agnostic and rule based, sparse, with interactions.
- Graph LIME: Interpretable model for graph networks from N-hop neighbourhood.
- XGNN: Post-hoc global-level explanations for graph neural networks.
- Shap Flow: Use graph-like dependency structure between variables

Based on each trained ML model type (e.g., Black-box, White-box), its input data (e.g., Tabular, image) and metadata, corresponding XAI techniques will be mapped, subsequently generating a containerized ***XAI explainer***, which latter will be called for ***XAI core service*** building contexts and delivering explanation for end-user. Figure 27 summarizes various XAI techniques with their inputs and outputs.

5 Data Security, Privacy, Traceability & Sharing

Task 2.1 starts the effort of establishing a resilient and trustworthy storage infrastructure within the AgriDataValue platform. To this end, a comprehensive approach is pursued to accommodate the diverse nature of data originating from IoT sensors, drones, and Earth Observation hubs. This task extends the storage functionality by encompassing not only data but also knowledge repositories in the form of Machine Learning (ML) models and smart contracts. A significant aspect of implementing a trustworthy storage and sharing system in Task 2.1 lies the incorporation of Blockchain mechanisms into the storage infrastructure. This integration aims to fortify the security, integrity, and privacy of the stored data and knowledge.

The interoperability of the developed storage infrastructure is one more aspect that is covered by Task 2.1. Here we ensure compatibility with existing Blockchain technologies, e.g. Ethereum. Additionally, the integration aspect includes implementing Gaia-X compliant gateways and IDS connectors, facilitating seamless interaction with external data sources. Lastly, to uphold principles of privacy by design, Task 2.1 explores advanced cryptographic mechanisms such as Attribute-Based Encryption (ABE), including CP-ABE and KP-ABE, alongside innovative signature schemes. This combination aims to guarantee data confidentiality and, where applicable, enforce time expiration.

The partners involved in developing the Data Security, Privacy, Traceability & Sharing component of the AgriDataValue platform will place particular emphasis on compliance with data protection regulations, with a focus on GDPR, and incorporate ethical considerations to maintain the latest standards of data privacy and security. By achieving these milestones, in Task 2.1 we will contribute significantly to the overarching goal of establishing a secure and interoperable data storage solution within the AgriDataValue platform.

5.1 Trustworthy Data and ML models storage and sharing (SECURESTORE)

5.1.1 Description

The SECURESTORE component serves as a trustworthy storage element within the AgriDataValue ecosystem, designed to provide a secure and organized storage solution. At its core is the integration of Minio storage, featuring various buckets tailored for multiple data types and sources. This structure facilitates the storage of data, ML models, and corresponding explanations, allowing efficient retrieval and management.

Within SECURESTORE, data originating from diverse sources like IoT sensors, drones, and Earth Observation (EO) hubs converted to follow the ADV data model will be classified and stored in designated buckets based on their type and source, ensuring a streamlined repository. In addition to data, SECURESTORE dedicates specific buckets for the storage of ML models. This approach will encompass not only the models themselves but also their associated metadata. Furthermore, SECURESTORE incorporates buckets for storing explanations linked to the stored ML models. These explanations, enriched with metadata, will contribute to an enhanced understanding of the rationale behind model predictions.

Integral to the seamless functioning of SECURESTORE is its interaction with the DKM component. The DKM component takes on the responsibility of partly populating SECURESTORE by inserting ML models, metadata, and explanations. This symbiotic relationship ensures a continuous and harmonious flow of information. The other component involved in populating SECURESTORE, is the STORE component, where the raw data coming from the pilots will be initially stored and aggregated before being moved for long term trustworthy storage in SECURESTORE. One of the main novelties in the ADV platform is that SECURESTORE goes beyond

conventional storage paradigms by integrating with blockchain technology. Smart contracts will be triggered post-insertion events by the SECURESTORE component. These contracts, executed on the blockchain, imprint essential information such as hash values and URLs, creating an immutable and transparent ledger of data, ML models, and associated explanations.

In terms of implementation, SECURESTORE involves the configuration of the Minio storage system inside a dockerised container, the creation of interfaces for DKM and STORE interaction, developing smart contracts for blockchain integration, and establishing of robust data flow management mechanisms. Security measures are also put in place, including encryption, access controls, and monitoring to safeguard the integrity and confidentiality of stored information.

The SECURESTORE component, through its organization, interaction with the DKM component, and integration with blockchain, realizes a secure, transparent, and interoperable storage solution within the ADV ecosystem.

5.1.2 Development view

5.1.2.1 Component diagram

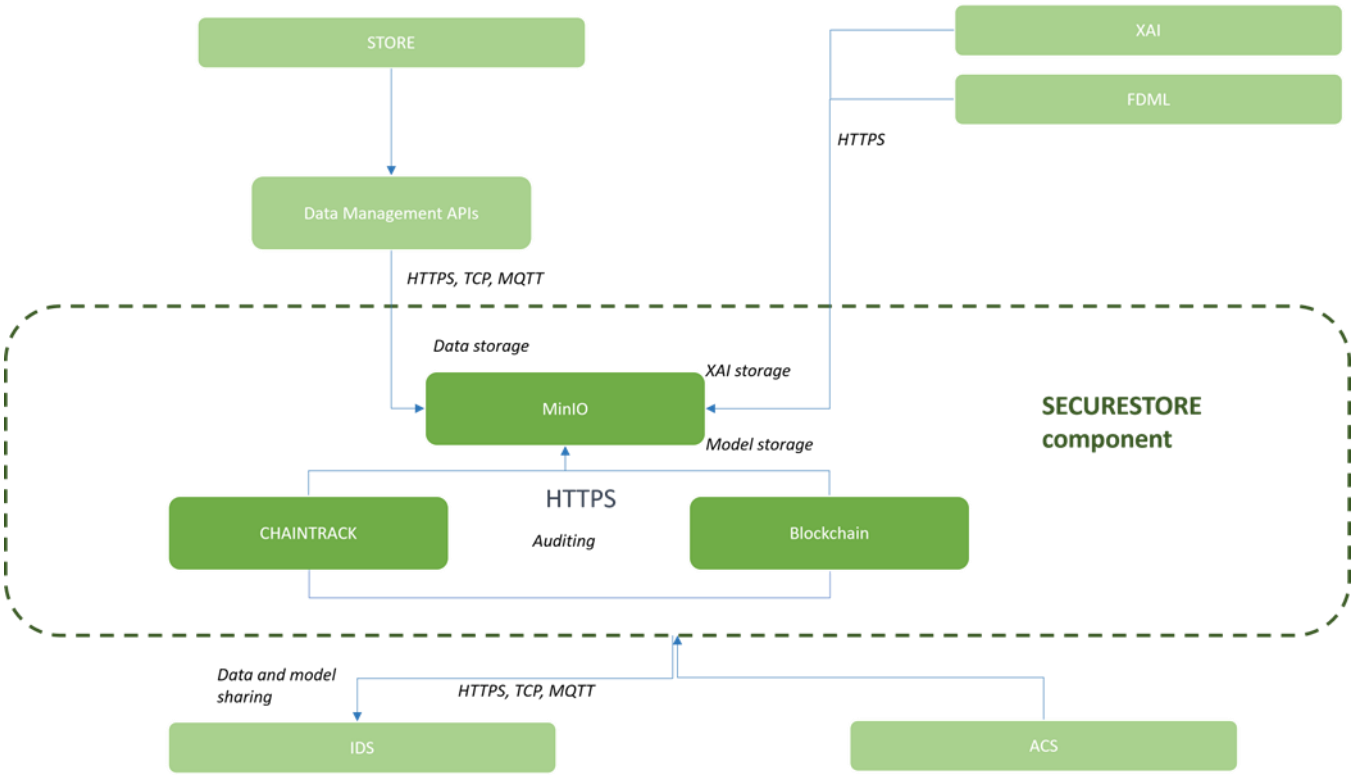


Figure 28. Logica view diagram for SECURESTORE

5.1.2.2 Building blocks

The SECURESTORE component of ADV contains the following sub-modules:

1. The actual storage system based on MinIO;
2. The CHAINTRACK component;
3. The Blockchain component storing smart contracts.

5.1.3 Process view

5.1.3.1 Sequence diagram

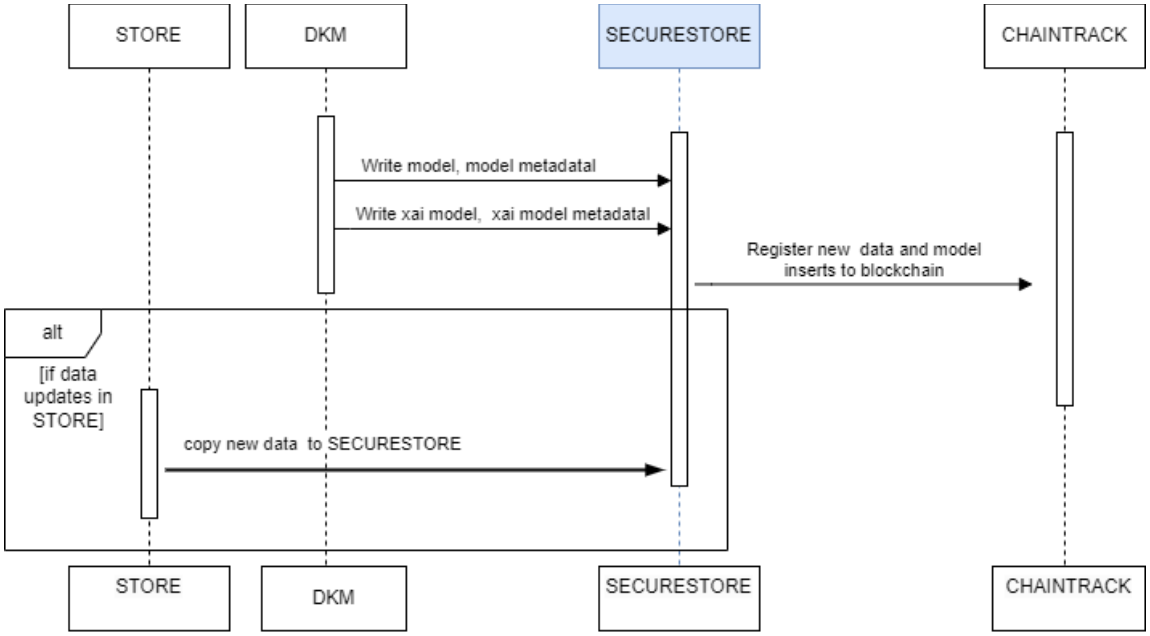


Figure 29. Sequence diagram for SECURESTORE

In the operational workflow of SECURESTORE (Figure 29), a periodic script is activated to monitor updates in the STORE data. Upon detecting changes, the script orchestrates the copying of new data to designated Minio buckets within SECURESTORE storage. Simultaneously, the DKM component contributes to this process by inserting FDML, XAI models, and their corresponding metadata to the SECURESTORE minion storage specific buckets, encapsulating the functional and explanatory aspects of the models. Within SECURESTORE, a cyclically running script takes charge of triggering a smart contract on the CHAINTRACK blockchain. The invoked smart contract, upon execution, performs the registration of new data and model insertions to the CHAINTRACK blockchain. This engagement with the blockchain ecosystem establishes a concrete link between the newly introduced data and models and their corresponding URLs, which are recorded on the CHAINTRACK blockchain for future reference.

5.1.4 Interfaces

5.1.4.1 Data models used in interfaces

Name	*AIM data model*	
Property	**Type**	**Description**
	Sensor, drone and EO data	All data inserted to SECURESTORE will follow the AIM model as formatted in the STORE component

Name	*FDML data model*	
Property	**Type**	**Description**

	FDML model data and associated metadata	FDML models and associated metadata to be inserted in the specific MinIO buckets

Name	*XAI data model*	
Property	**Type**	**Description**
	XAI model data and associated metadata	XAI models and associated metadata to be inserted in the specific MinIO buckets

5.1.4.2 Description of APIs

In the current implementation of the deployed SECURESTORE component access control will be done based only on credentials, in the following iterations there will be a Keycloack integration with the minIO storage.

(REST)

Title	*This field holds the description of the API*
URL: *This field holds the relative path to the described API. For simplicity Root path can be cut off from this description and can be placed as a hypertext above the API template*	
/minio-server:9000	
Method *This field holds the type of the Method used*	
GET | POST | DELETE | PUT	
Data Params *This field holds the body payload of a get/ post request.*	
Required:	
endPoint =[alphanumeric]	*minio server url*
accessKey=[alphanumeric]	*Credentials to access minio storage*
secretKey=[alphanumeric]	*Credentials to access minio storage*
Optional:	
bucketname=[alphanumeric]	*minio bucket to be accessed*
Success response <What should the status code be on success and is there any returned data? This is useful when people need to know what their callbacks should expect>	
200 OK 204 No content Content: { }	*Both codes indicate success. In the case of a successful upload an empty response will be returned.*
Error response *This field holds the list of all possible error responses. Doing that, helps prevent assumptions of why the endpoint fails and saves a lot of time during the integration process.*	
4xx (client errors) 5xx (server errors) minio specific: “NoSuchKey”	standard http error responses or minio specific error in xml format
Sample call *This field holds a possible sample call to the described endpoint in a curl-like format. Please, choose the format wisely so that is clear and easy to read by the interested parties.*	
curl -X GET \ -H "Authorization: AWS ACCESS_KEY:SECRET_KEY" \ http://your-minio-endpoint:9000/your-bucket-name	
Notes *This field holds any additional helpful info related to this endpoint.*	

5.1.5 Technologies and implementation details

The current stage of the SECURESTORE component has implemented a containerized Minio storage in a Docker.

The implementation of Dockerised Minio Storage within SECURESTORE involved the use of containerization technology to encapsulate the Minio storage system, providing a lightweight, scalable, and portable solution. Some implementation details include:

- **Docker** is employed to containerize the Minio storage system. Containers encapsulate Minio and its dependencies, ensuring consistent and reproducible deployment across various environments.
- **A Minio Docker image** sourced from the official Minio Docker image available on Docker Hub. This image includes the Minio server and provides a baseline for container instantiation.
- **Configuration settings for Minio**, such as access keys, secret keys, and storage parameters, are managed using environment variables. This allows for easy customization without modifying the underlying Docker image.
- Persistent storage is achieved using **Docker volumes** mounted to the Minio container, ensuring that data persists even if the container is stopped or removed.

5.2 DLT-based supply chain tracking solution (CHAINTRACK)

5.2.1 Description

Ensuring the safety, quality, and authenticity of food products heavily relies on the traceability of the food supply chain. As consumers become increasingly concerned about the origin and sustainability of the products they consume, the need for innovative technological solutions to address these challenges becomes evident. Blockchain technology emerges as a promising solution to enhance the traceability of agri-food supply chains, offering significant advantages from both functional and technical perspectives.

Blockchain serves as an immutable and shared record of all transactions and events occurring along the agri-food supply chain. This characteristic empowers consumers with access to comprehensive **information** about the entire production process, from planting to distribution, ensuring **transparency** and **verifiability** of information. Moreover, the blockchain enables a permanent record of information pertaining to the provenance and **authenticity** of food products, rendering it impossible to counterfeit or falsify products. But a Blockchain-based solution also enhances **food safety**, as rapid detection of problems or contamination of food products becomes possible due to the accurate traceability provided by the technology, which protects consumers from potential **health risks**.

A Blockchain solution then simplifies the monitoring of activities along the supply chain, enabling supply chain actors to identify any inefficiencies or delays early and make corrections proactively, leading to greater **efficiency** and reduced waste.

Blockchain uses advanced **cryptography** and a distributed structure to make data immutable and safe from tampering. This ensures that traceability information is reliable and accurately stored.

The **decentralized** nature of the Blockchain allows information to be shared among all authorized participants in the agribusiness supply chain and all actors in the supply chain to be identified. Moreover, the Blockchain-based solution simplifies process management, reducing the need for manual checks and paperwork resulting in faster and more accurate operations.

Blockchain technology can be adapted to meet the needs of agrifood supply chains of different sizes and complexity. In addition, interoperability between different Blockchain and information systems allows for greater collaboration between different actors within the supply chain.

The CHAINTRACK component provides the service of tracking of supply chains of various agricultural products. It is a development based on Ethereum and Hyperledger Besu technologies.

5.2.2 Development view

5.2.2.1 Component diagram

The CHAINTRACK component includes a REST API that interfaces the Blockchain Network and the Store component. Information about a particular supply chain is registered in the SECURESTORE and bound with an account on the blockchain network.

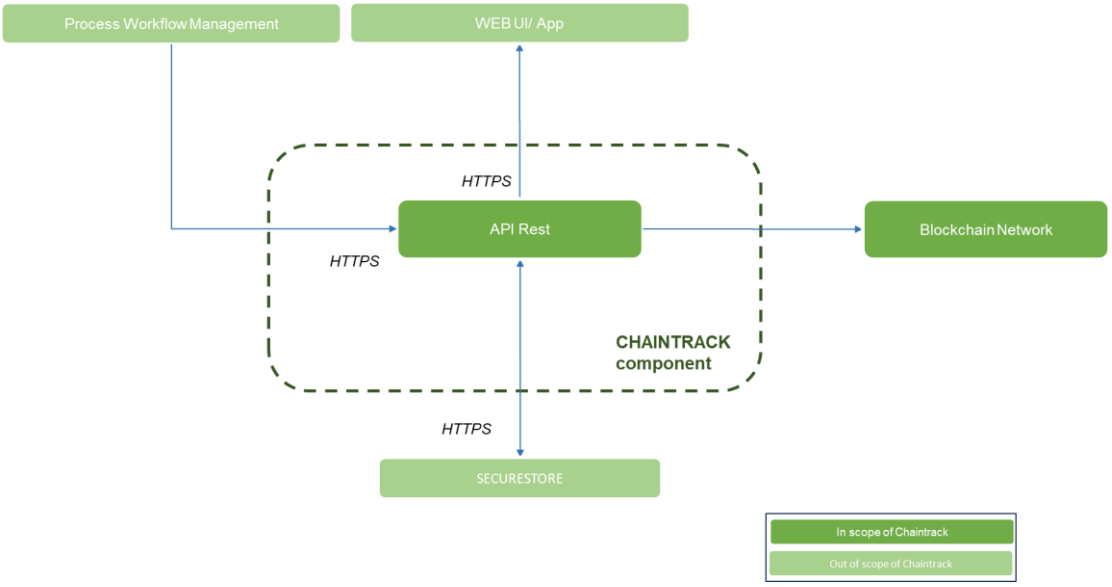


Figure 30: Logical view diagram – CHAINTRACK

Supply chain process data coming from workflow management is saved on the blockchain network in a hashed format. Actual data is saved in the SECURESTORE[41] associated with the relevant hash. Retrieval of process information from the UI/APP is directed to SECURESTORE; information and the corresponding hashed version are returned. The transaction data can on-demand be verified against the blockchain, by comparing the hash with the one returned from the blockchain.

5.2.2.2 Building blocks

The Figure 31 below describes the deployment architecture of the CHAINTRACK module highlighting the communication flows in place between two available technical interfaces. Besides the REST API, that provides a comfortable integration channel for web-based clients, it is also possible for an external actor to interact directly with the blockchain network.

[41] Along this section 5.2 the inner composition, implementation, and behaviour of the SECURSTORE component is not discussed. The SECURESTORE is intended as a persistence service layer accessible to the CHAINTRACK application, sometimes alternatively referred to as “the database”. For detailed specifications of the SECURESTORE component and its interfaces please refer to section **Error! Reference source not found.**.

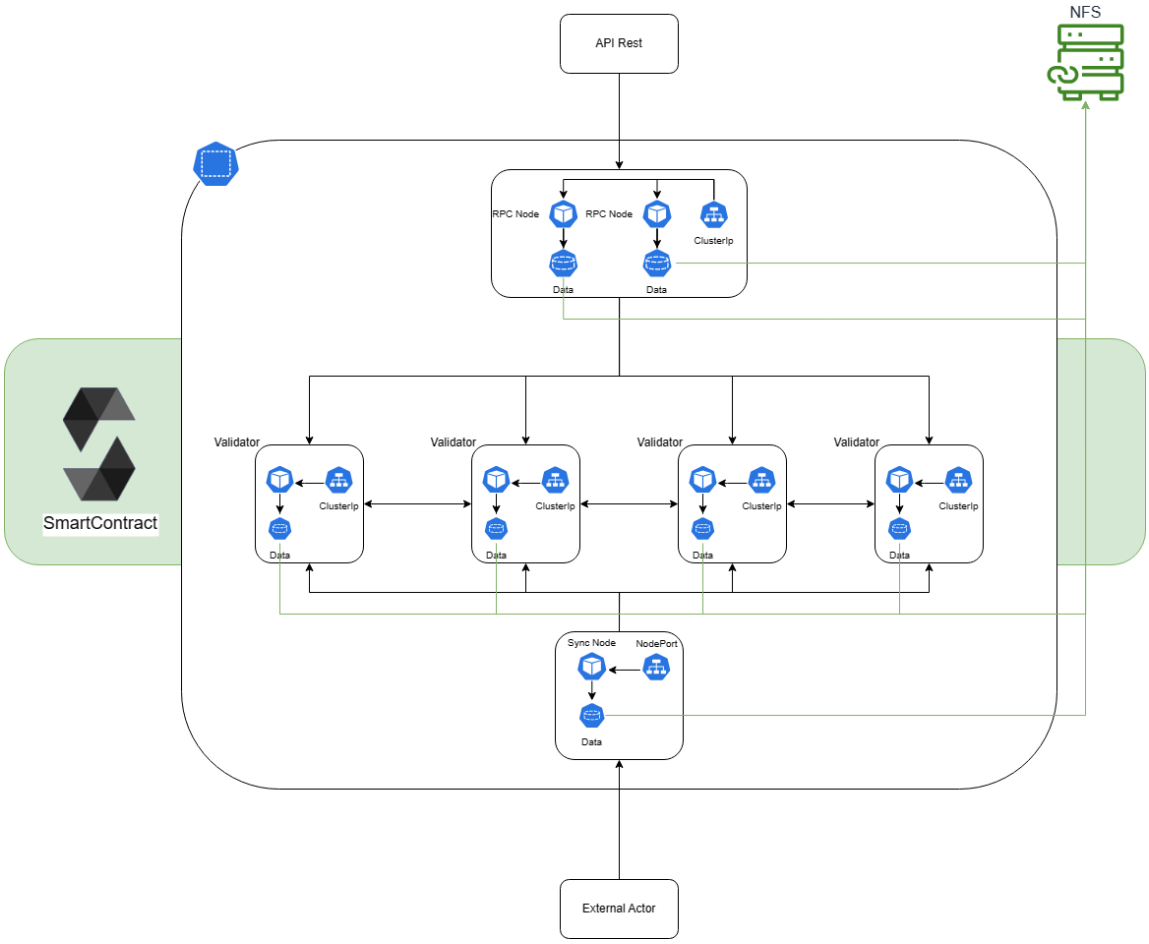


Figure 31: CHAINTRACK deployment architecture.

The secure storage is achieved through the combined action of two sub-components: an object store for the actual conservation of the data (SECURESTORE component) and a blockchain network to certify in a non-disputable and immutable way the relevant steps of the data lifecycle.

5.2.3 Process view

The application process put in place for tracking supply chain information is strongly dependent on the supply chain use case. In general, the information entities to record, track, and certify production and distribution steps differ according to the specificity of the production/supply chain being tracked. The particularity of the supply chain impacts the definition of the data model in the storage component and, as a side-effect, the structure of the REST APIs methods input/output data. It is possible, for very standardized productive processes, to make an attempt of generalizing such model, at least at the level of market sector and product type but, unless the process is very well regulated at administrative level (e.g. the DOCG certification process for a selected production chain), it is expected that particular producers and consortia had developed their own production/distribution processes, targeting specific and differentiating business plans and KPIs. Therefore, the need to instantiate specific developments in terms of building a supply-chain-dependant database schemas and REST interfaces must be taken into account when approaching such a development. The overall set-up process is well represented in Figure 32, Outlining how to commission a DLT-based supply -chain tracker into service. Specialized configuration actions need to be done by the tracker's owner normally translating into the need to develop a dedicated database schema and corresponding set of APIs.

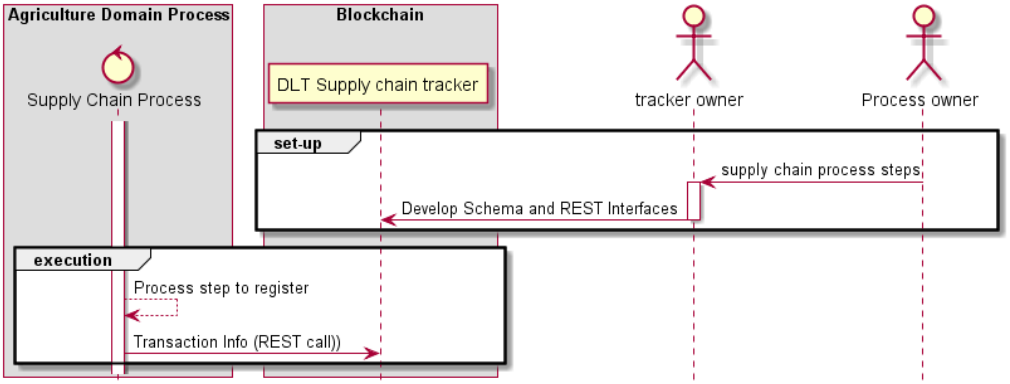


Figure 32:Development usually needed to configure the supply chain to track.

At this level the interaction between the supply chain process and the AgriDataValue platform of platform can be appreciated in terms of general services. At the time of this document's authoring, no specific low- level requirements addressing supply chain operations have been defined, hence, it is not yet possible to give a full definition of the REST interfaces now of the data model, for the aforesaid reasons.

Nevertheless, to allow a deeper understanding of the interactions and the actual capabilities of the solution in a real-life scenario, the decision was made to implement a specific example as a "reference" supply-chain use-case. This realization, while constituting a synthetic example and therefore passible of variations or future reconsiderations, is expected however be useful to serve for the first infrastructural end-to-end validation test cases.
The example, dealing with the **traceability of an olive oil production chain** accomplished via the utilisation of a Blockchain solution, is a process that involves several actors along the entire supply chain, from production to distribution of the final product. The various phases of this process are detailed in the next paragraphs by a UML representation of the relevant steps.

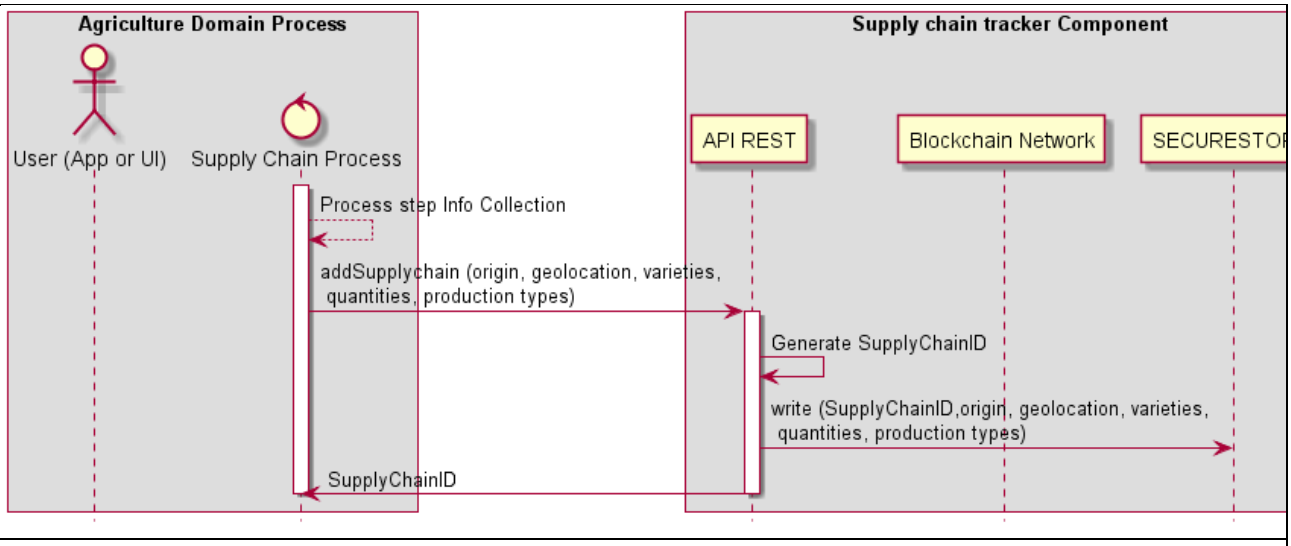
5.2.3.1 Step 1: Collection

The flow begins with production and, therefore, with the **collection of information** about the place of origin of the raw "material" (olives):

- Origin
- the place of production (geolocation)
- the methods of cultivation or extraction
- varieties
- quantity

Table 5: CHAINTRACK for Olive Oil: "Collection" technical steps

What happens in the background



The information about the supply chain is written in the database for reference. No information is written on the blockchain. A supplyChainID is generated to bind info and later to retrieve it

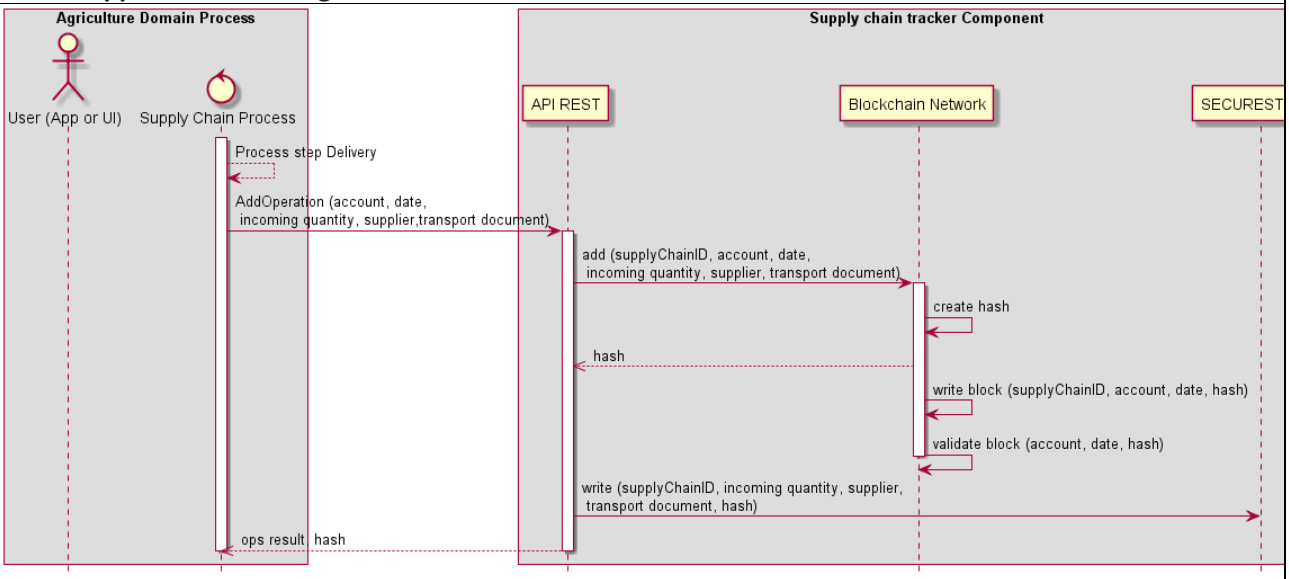
5.2.3.2 Step 2: Delivery to the mills

Subsequently, the **delivery of the olives to the mill** takes place, in this phase the data involved are:

- Company
- Date of operation
- Incoming quantity
- Unit of measure
- Supplier
- Delivery slip

Table 6: CHAINTRACK for Olive Oil: "Delivery" technical steps

What happens in the background



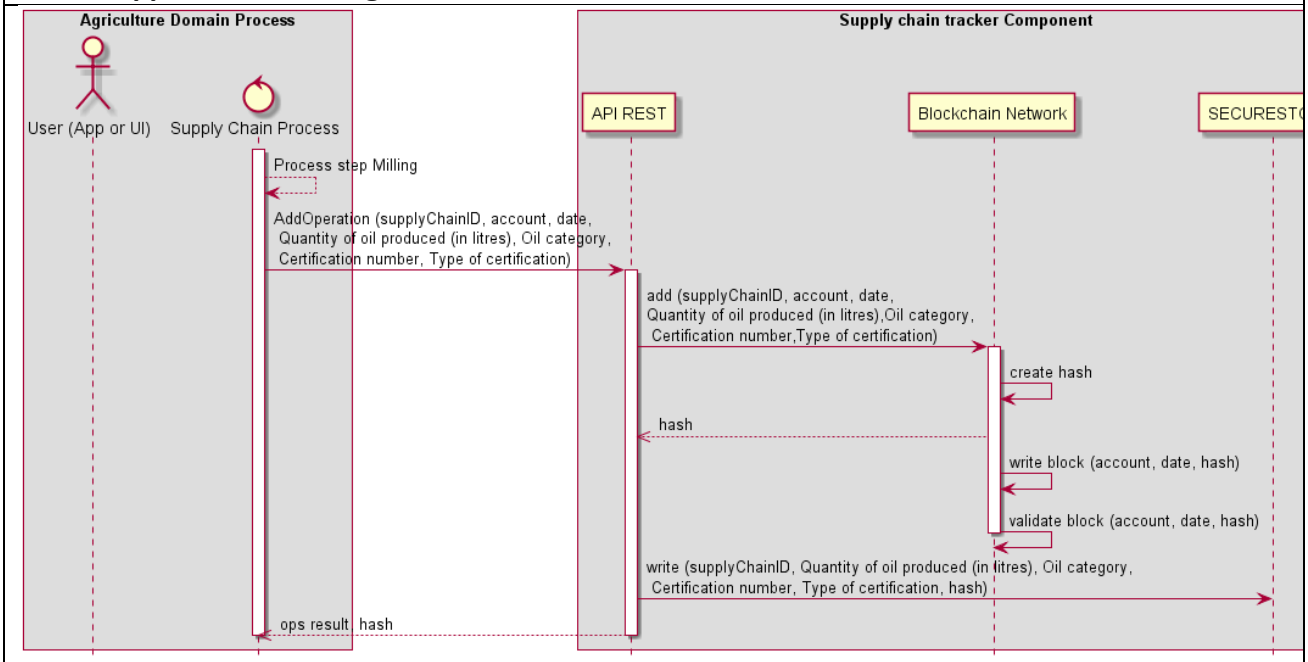
The information about the supply chain is written in the database. The blockchain stores the hashed information.

5.2.3.3 Step 3: Milling

The flow continues with the **milling** (processing), the data concerned are:

- Company
- Date of operation
- Quantity of oil produced (in litres)
- Oil category
- Certification number
- Type of certification

Table 7:CHAINTRACK for Olive Oil: "Milling" technical steps

What happens in the background

The information about the supply chain is written in the database. The blockchain stores the hashed information.

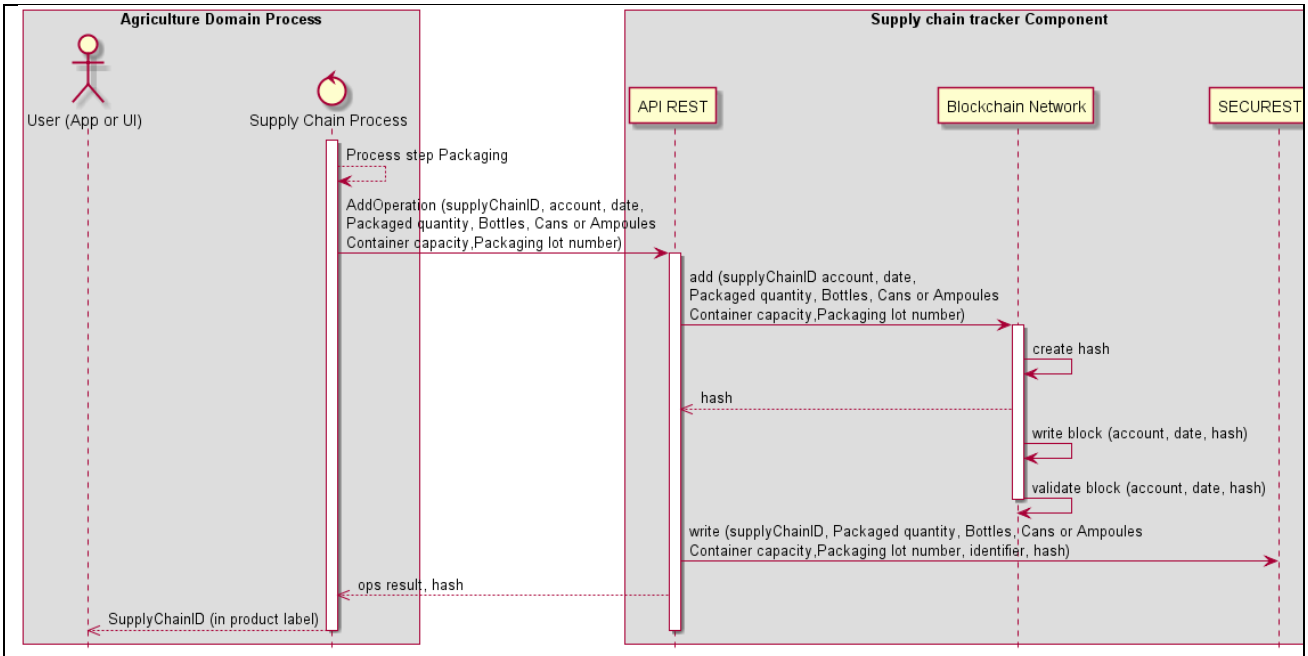
5.2.3.4 Step 4: Packaging

Then **packaging** takes place with labelling via QR Code or NFC, the data collected are:

- Company
- Date of operation
- Packaged quantity
- Unit of measurement lt,hl
- Bottles, Cans or Ampoules
- Container capacity
- Packaging lot number

Table 8: CHAINTRACK for Olive Oil: "Packaging" technical steps

What happens in the background



The information about the supply chain is written in the database. The blockchain stores the hashed information. The label of the finished product contains a reference to the supplyChainID (QR Code or NFC)

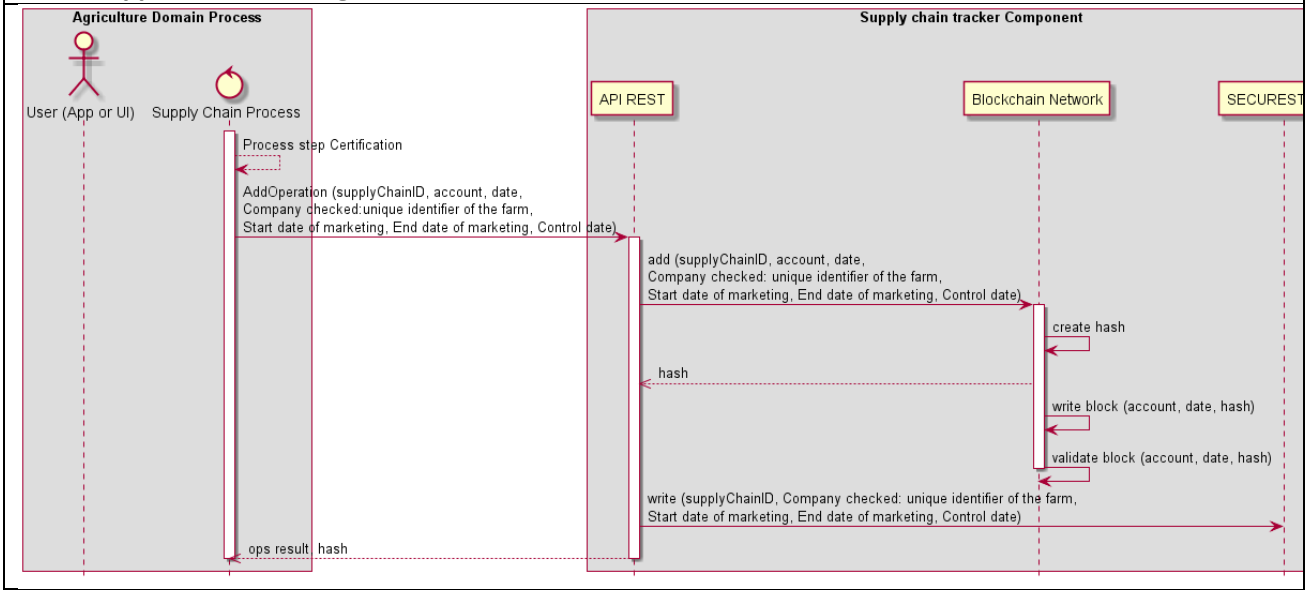
5.2.3.5 Step 5: Certification

Immediately afterwards, the **certification** phase takes place and the data concerned are as follows:

- Company
- Date of operation
- Company checked: unique identifier of the farm
- Date of start of marketing
- End date of marketing
- Control date

Table 9: CHAINTRACK for Olive Oil: "Certification" technical steps

What happens in the background



The information about the supply chain is written in the database. The blockchain stores the hashed information.

5.2.3.6 Step 6: Distribution

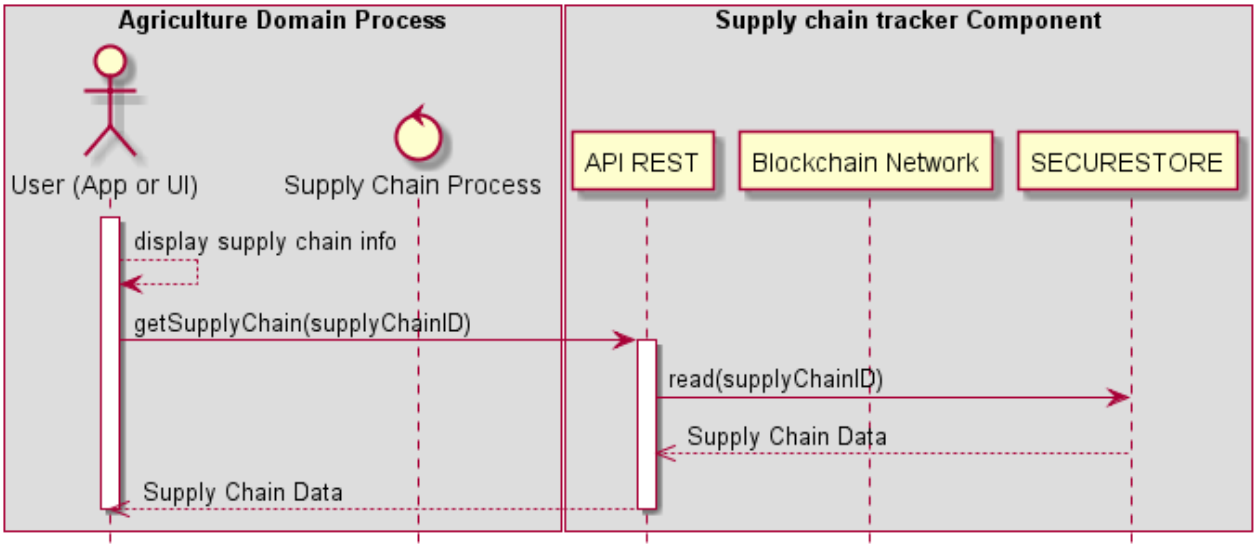
Finally, there are two more steps:

- distribution (at the point of sale and the final consumer);
- other relevant information regarding the quality and sustainability of the product.

All this information is organized step by step into blocks of data (this is specific information, such as a batch of oil produced on a specific date or at a specific location):

- Each block in the Blockchain solution receives a unique identifier called a **"hash"**, which uniquely represents the information contained, and serves as a fingerprint to ensure the integrity and authenticity of the data.
- Next, the blocks containing the oil information are **validated and verified** by the Blockchain network nodes, which can be operated by different participants in the supply chain, such as producers, suppliers, transporters, and distributors.
- Once validated, each block is added to the chain, creating a sequential and **chronologically ordered record** of all steps in the oil production and distribution process.
- By the time the oil reaches the final consumer, the consumer can access the oil's traceability information using the code or tag on the product label. The consumer can easily **verify** the authenticity of the oil, know its origin, and assess the sustainability of the production practices adopted.

Table 10: CHAINTRACK for Olive Oil: "Distribution" technical steps

What happens in the background

The user checks the info about the supply chain via UI or app providing the supplyChainID as input (QR Code or NFC taken from the product label). The whole supply chain data is retrieved from the database and returned. Each step can be object of further verification by querying the relevant transaction on the blockchain via dedicated APIs calls (not shown in the diagram).

5.2.4 Interfaces

The Interfaces activated in the process described above are here specified from a technical point of view.

It should be noted that both the model and the interfaces contain input/output elements not strictly defined, since a general model/interface featuring all the possible processes has not been developed yet.

Simultaneously, the provided tables put the accent on relevant parameters that every product/process owner is expected to provide, before the next analysis/development phases are initiated.

A specific interface definition corresponding to the "reference" Olive Oil Supply chain tracking use cases will be provided as part of the component validation kit.

5.2.4.1 Data models used in interfaces

Name	CHAINTRACK Data model	
Property	Type	Description
Property	**Type**	**Description**
message	string	Generic message
messageObject	object	Message Response from server
response	object	Success message from server
notFoundResponse	object	Not found message from server
unauthorizedMessage	object	Unauthorized message from server
loginWeb	object	Payload for login
supplyChain	object	Payload representing a supply chain
operation	object	Payload representing an operation in the supply chain
updateOperation	object	Payload representing an update for a specific operation in the supply chain
findShapeFileBody	object	Payload for a request to search a POI related to an actor in the supply chain
signedOperation	object	Payload to add a new operation in the supply chain
signedSupplyChain	object	Payload to add a new supply chain in the application
deleteSupplyChainBody	object	Payload to delete a supply chain
accountETH	string	Ethereum account who isf performing the API operation
operationList	object	List of operations inside a supply chain
actorEnrollment	object	Payload to finalize the enrolment of a user
actorEnrollmentPending	object	Payload to start the enrolment of a user
certifiedOnBlockchainBody	object	Payload to certify supply chain's operations to blockchain
addMultimediaBody	object	Payload to add multimedia resource related to a company in the supply chain
findMultimediaBody	object	Payload to search multimedia resources related to a company in the supply chain
multimediaElement	object	Payload representing a multimedia resource related to a company in the supply chain
addSupplyChainBody	object	Payload to create a new supply chain
addSupplyChainDraftBody	object	Payload to create a new supply chain in draft status
addOperationBody	object	Payload to save a new operation for a supply chain
addOperationDraftBody	object	Payload to save a new operation in draft status for a supply chain
detailOperationBody	object	Payload to get details for a specific operation in a supply chain
addRemoveActorBody	object	Payload to add/remove an actor in a supply chain
getOperationListBody	object	Payload to get the list of operations in a supply chain
deleteOperationInDraftBody	object	Payload to delete draft operation in a supply chain

5.2.4.2 Description of APIs

Title	**Retrieves the dynamic template to customize the dashboard**
URL: /getTemplate	
Method	
POST	
Data Params	
Required:	
accountETH=[string]	*Ethereum account calling the service*
Success response	
200 Content: { }	*The template to be used in the front-end dashboard*
Error response	
401 Content**:** {"message": "string"}	*Not authorized*
404 Content: “message-string”	*Not found*
500 Content: “message-string”	*Internal server error*
501 Content: {"message": "string"}	*Request not managed by the system*
Sample curl --location --request POST 'http://localhost:3000/getTemplate' \ --header 'Content-Type: application/json' \ --header 'Accept: */*' \ --header 'authorization: {{apiKey}}' \ --data-raw '{ "accountETH": "aliqua aute" }'	
Notes	

Title	**Returns the companies and the employees associated with it, if any**
URL: /companies/{accountETH}	
Method	
GET	
URL Params	
Required:	
accountETH=[string]	*The Ethereum address of the user performing the operation*
Optional:	
cuaa=[string]	*The cuaa of the company itself or the one of the related CA*
commodity_code=[string]	*The commodity code related to the accounts of the company to search*
Success response	
200 Content: { }	*The company with its employees*
Error response	
401 Content**:** {"message": "string"}	*Not authorized*
406 Content: “message-string”	*Account cannot enrol employees*

500 Content: “message-string”	*Internal server error*
Sample curl --location --request GET 'http://localhost:3000/companies/non?cuaa=non&commodity_code =non' \ --header 'Accept: */*' \ --header 'authorization: {{apiKey}}'	
Notes	

5.2.5 Technologies and implementation details

The technologies that will be used for the blockchain are Ethereum and Hyperledger Besu, while for the Storage, MinIO and Python are expected to be used as parts of the SECURESTORE component.

Component	Storage Type
Blockchain Network	NFS Server
API-Rest	(SECURESTORE)

For the API-Rest component the NodeJS and Express framework have been used to expose the API. It will use the SECURE STORE as the persistence layer. For the blockchain an NFS Server has been used to store the persistent files related to the peers that compose the network.

5.3 Access Control System (ACS)

5.3.1 Description

The Access Control System (ACS) tool offers access rights and a dependable authentication system. Among the features that the ACS tool supports are Authentication, Authorization, and Accounting (AAA) services. The purpose of the ACS tool is to authenticate users or components trying to access AgriDataValue resources. After that, it ensures that they can only use AgriDataValue resources for which the required authorizations have been specifically given.

The ACS tool has embraced the following technologies: SAML 2.0, OpenID Connect, and OAuth 2.0.

OAuth 2.0 authorization framework [11] allows a user to grant a third-party website or application access to their protected resources without having to reveal their identity or login credentials. OAuth divides the responsibilities of the resource owner and client and offers an authorization layer. The client also requests access to resources that are hosted by the resource server and managed by the resource owner. The resource owner then provides the client with an alternate set of credentials. An access token, which is a string with specified scope, lifetime, and other access attributes indicated, is sent to the client. Access tokens are issued to third-party clients by an authorization server with resource owner consent. Next, the client uses the access token to gain access to the secured resources.

OpenID Connect (OIDC) [12] is an identity layer Developed on top of the OAuth 2.0 framework. It makes it possible for third-party apps to get basic user profile data and confirm the end user's identity. OIDC uses JSON web tokens (JWTs), which users can acquire through flows that follow OAuth 2.0 guidelines. OIDC is concerned with user authentication, whereas OAuth 2.0 deals with resource access and sharing. Its goal is to provide users with a single login to use on several websites. A user is redirected to the relevant OpenID site where

they are currently logged in whenever they need to use OIDC to log into a website. They are then taken to the website after that.

SAML 2.0 standard (Security Assertion Markup Language) [13] offers online business partners an XML-based framework for exchanging and describing security-related data. Applications that operate across security domain boundaries can trust the security information expressed in these portable SAML assertions. To generate, request, communicate, and use these SAML assertions, specific guidelines are outlined in the OASIS SAML standard. These portable SAML assertions express security information that can be trusted by applications operating across security domain boundaries. The OASIS SAML standard specifies precise guidelines that must be followed in order to create, request, communicate, and use these SAML assertions.

The AgriDataValue ACS tool is available at https://auth.platform.agridatavalue.eu/, it provides single sign-on functionality and is based on Keycloak (Keycloak, n.d.), which is supported by RedHat and provides an identity and access management solution for multiple applications and services. A connection between the AgriDataValue components and the ACS tool is intended and will be used for inter-component communications.

5.3.2 Development view

5.3.2.1 Component diagram

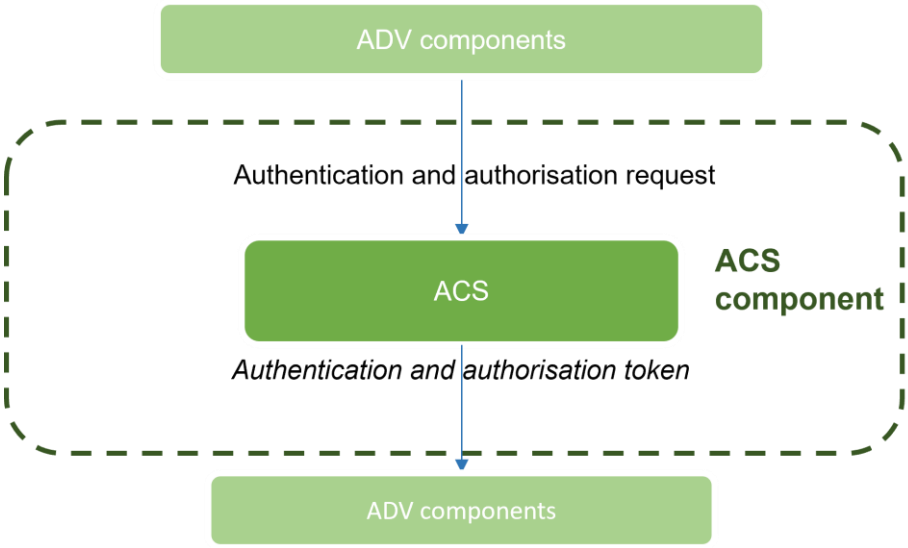


Figure 33: ACS - Component diagram

The diagram in Figure 33 depicts the positioning and the dependencies of the ACS component in relation to the rest of the platform's components. As described earlier it serves as an auth(z) service for the platform's components.

5.3.2.2 Building blocks

There are no internal building blocks of the ACS component.

5.3.3 Process view

5.3.3.1 Sequence diagram

Figure 34 depicts the process of a component requesting access to another component's resource from a high-level point of view. The authentication and authorisation process are managed through the ACS component which facilitates the access among components in the ADV platform and their access permissions.

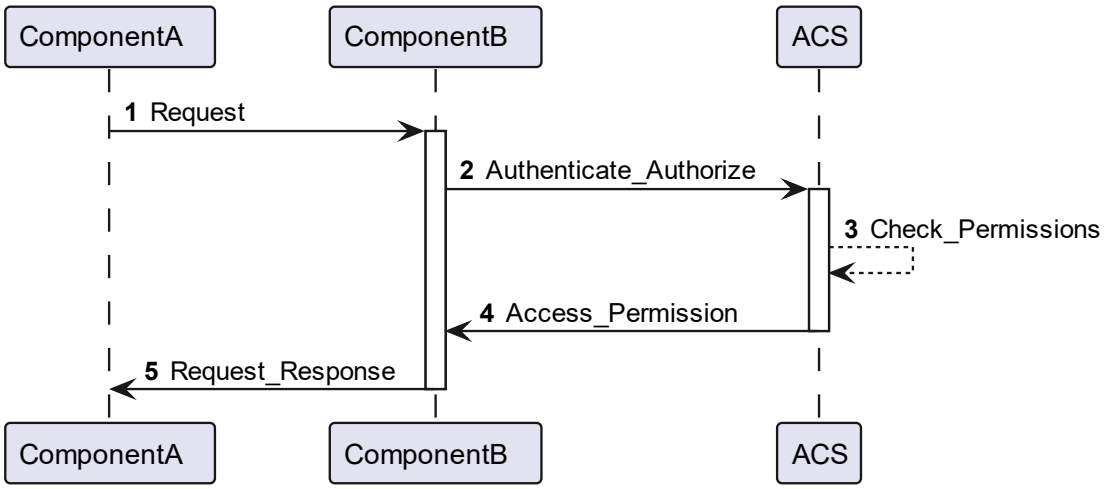


Figure 34: ACS - Sequence diagram

5.3.4 Interfaces

5.3.4.1 Data models used in interfaces

The ACS component does not use the ADV data model as it is based on Keycloak which is an external tool. Therefore, the ACS will use Keycloak's data model and interface as described in the documentation available online[42].

5.3.4.2 Description of APIs

The ACS API is merely Keycloak's API as described in Keycloak's online documentation.

5.3.5 Technologies and implementation details

As mentioned above, Keycloak has been used as the technology under the hood of the ACS component.

In the following screenshots the stages of the ACS service for granting access for the authorized user to the AgriDataValue resources are demonstrated.

[42] https://www.keycloak.org/documentation

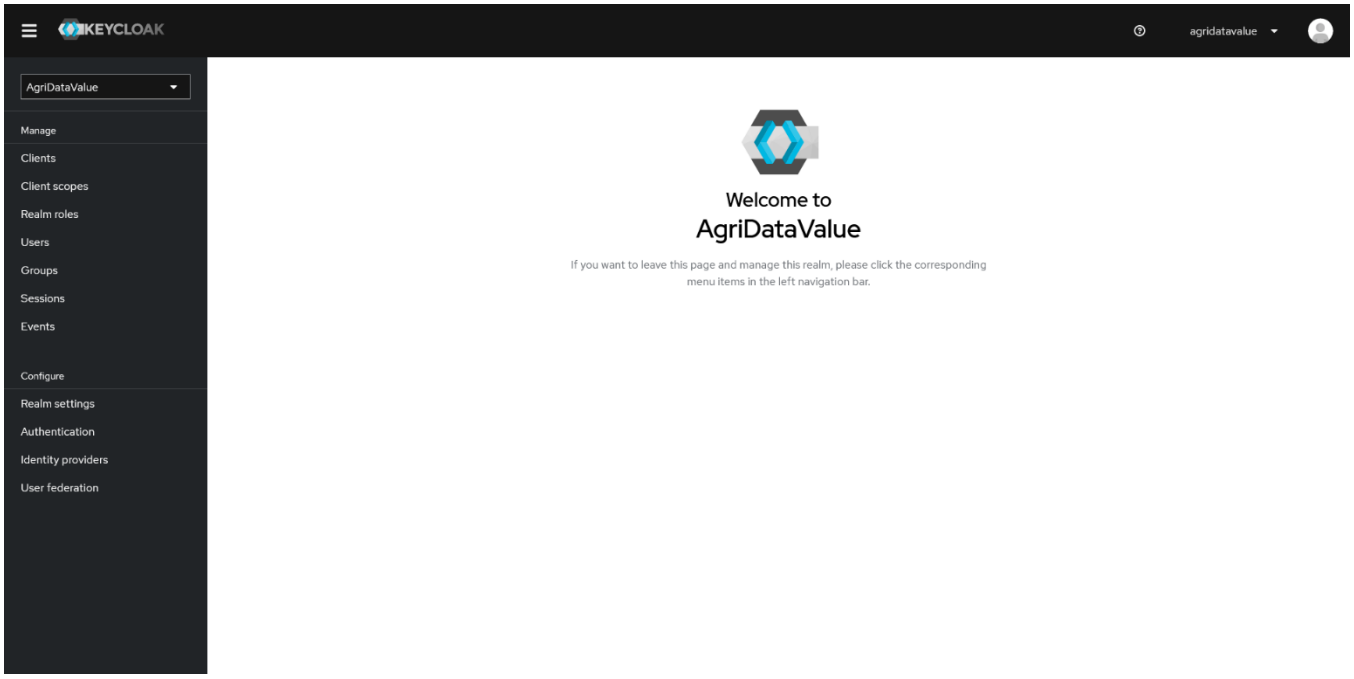


Figure 35: ADV ACS - Keycloak main page

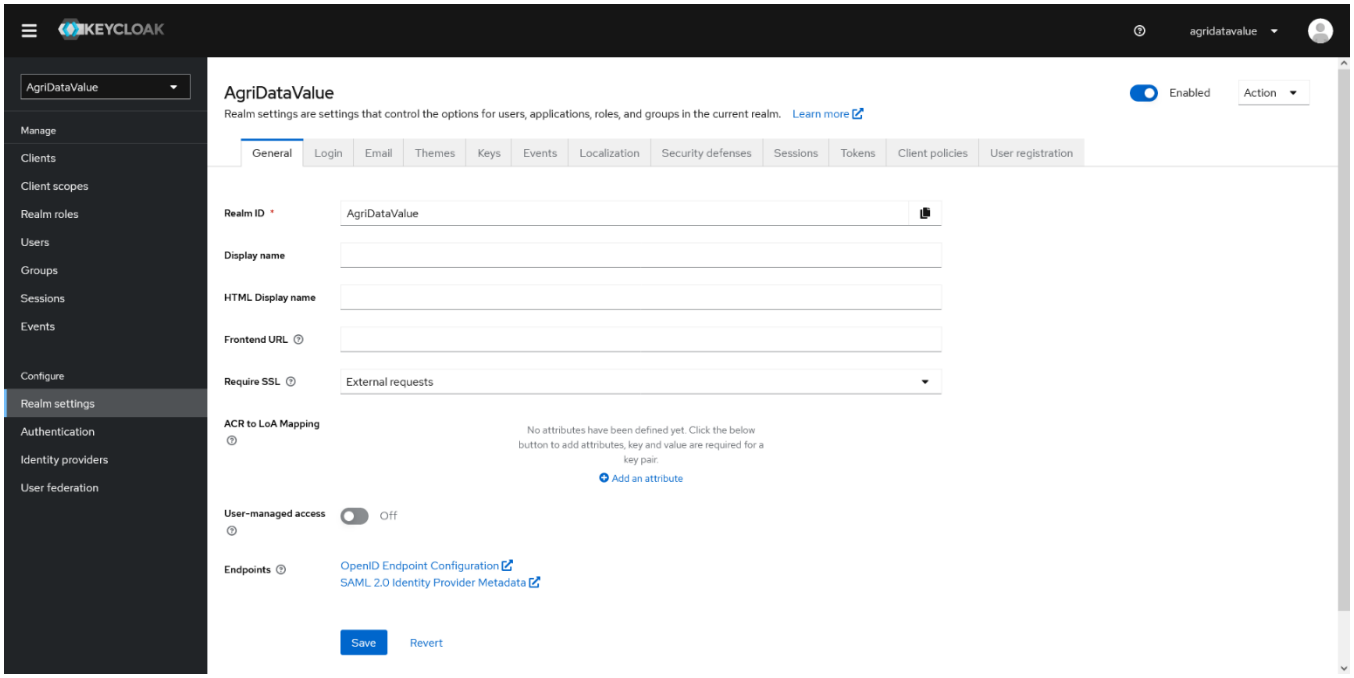


Figure 36: ADV ACS - Realm page

5.4 International Data Spaces (IDS) component(s)

5.4.1 Description

In the landscape of data sharing and interoperability, the International Data Spaces (IDS) framework emerges as a comprehensive solution, fostering secure and trustful relationships among diverse entities. The IDS Connector, key component within this framework, serves as a gateway for seamless data transactions, ensuring the integrity, confidentiality, and interoperability of information exchanged among participants. The IDS Connector Core Service embodies a spectrum of functionalities, from Authentication and Data Exchange

to Remote Attestation and Contract Management. Each component plays a crucial role in upholding the principles outlined in the IDS Reference Architecture Model (IDS RAM)[43], delineating the intricate web of interactions essential for reliable and secure data spaces. Furthermore, the internal sub-components/modules of the IDS Connector, such as the IDS Connector and Dataspace Protocol, provide a fair understanding of the technological implementations and protocols governing data spaces. With a diverse array of connectors and protocol options, including open-source solutions the IDS framework stands at the forefront of shaping a robust and flexible infrastructure that transcends technical specifications, encompassing organizational, legal, and trust-based considerations for secure and sovereign data exchange.

5.4.2 Development view

5.4.2.1 Component diagram

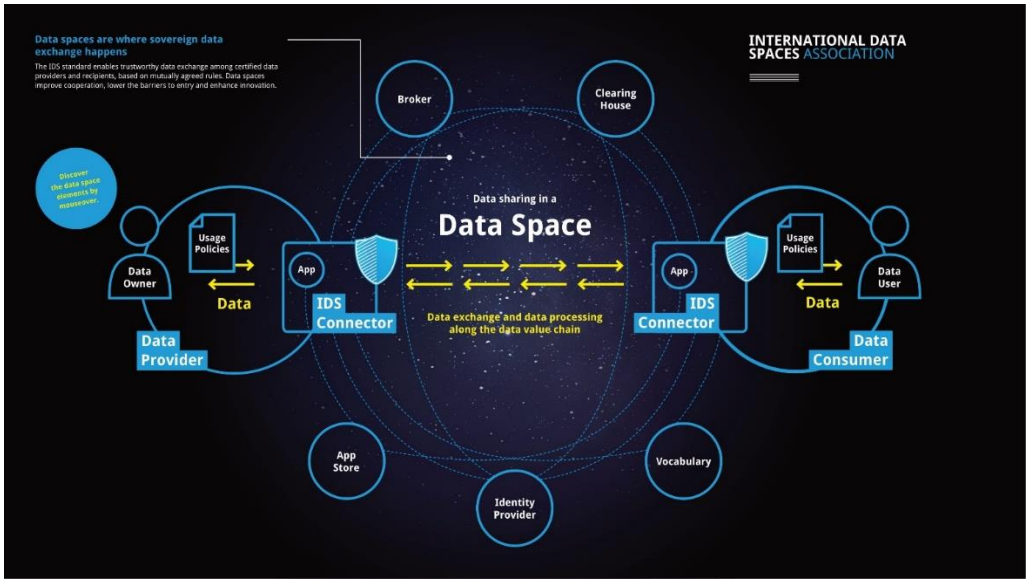


Figure 37: Visual description of a Data Space and IDS Components

[43] https://docs.internationaldataspaces.org/ids-knowledgebase/v/ids-ram-4/

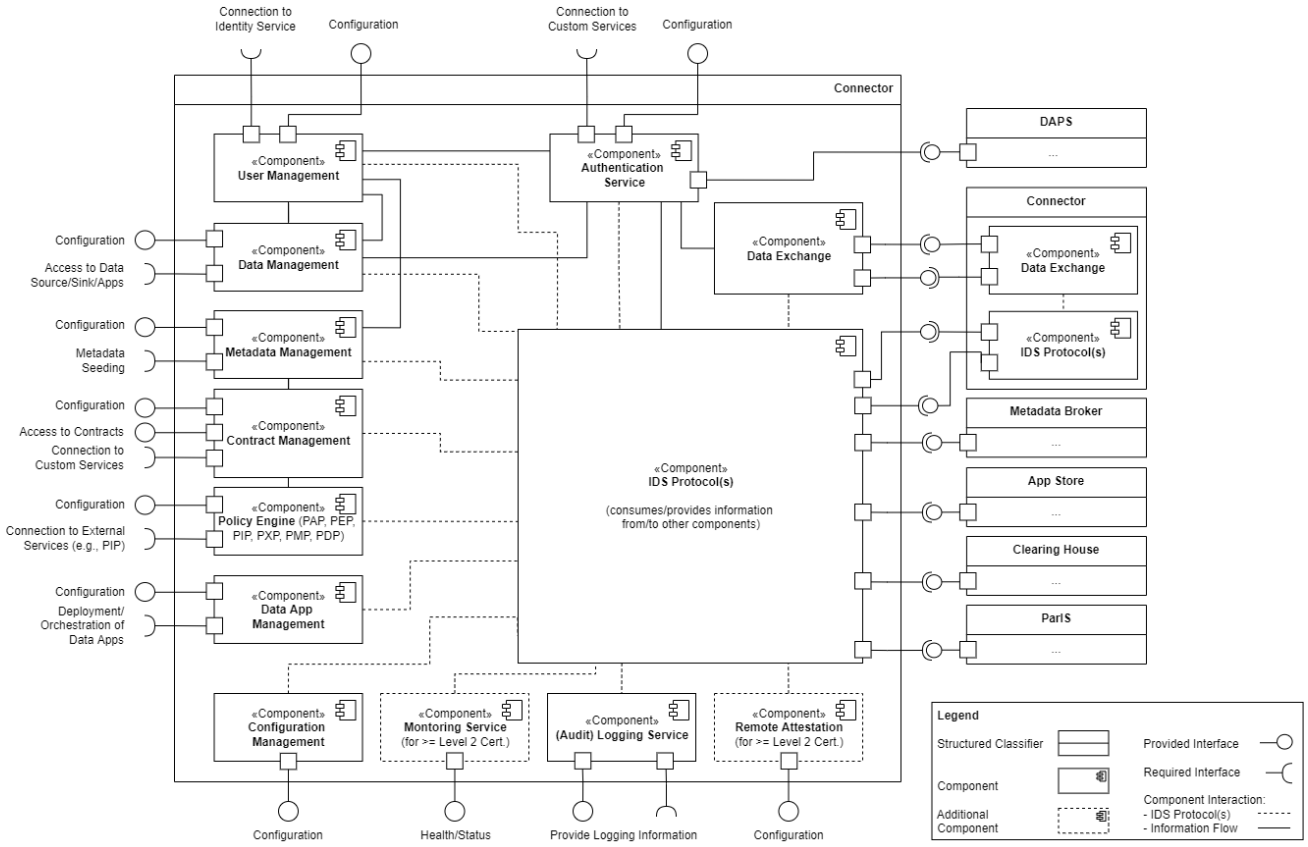


Figure 38: Functionalities of the IDS Connector Core Service(s)

Figure 38 above illustrates the distinct functionalities of the IDS Connector Core Service(s) through a UML deployment diagram, representing each function as a separate component. While the diagram purposefully outlines the components' external interfaces, it omits the internal ones due to their variability across different implementations. Additionally, to maintain clarity, not all component interactions are depicted.

Below is a description of each functionality:

- **Authentication Service:** Manages information essential for authenticating the IDS Connector with other backend systems or authorizing access between the IDS Connector and other IDS participants. It recommends a strict division of internal and external access credentials for security purposes. This service offers interfaces for both configuration and incorporation of custom authentication services. It maintains the Key/Trust Store for IDS Protocols, credentials for Data Management and Data Exchange with external systems, and access control information for IDS-related Data Exchange and Data Management. This arrangement is indicated by a solid line within the IDS Connector.
- **Data Exchange:** This component has interfaces essential for data sharing with other IDS Participants, capable of being hosted on different infrastructures from the IDS Protocol(s) component. It allows for multiple instances to accommodate various protocol bindings and does not handle IDS-specific interfaces or interpret the IDS Information Model.
- **IDS Protocol(s):** Facilitates at least one IDS-specific interface, as prescribed in IDS-G, to execute processes detailed in Section 3.4 of IDS Reference Architecture Model (IDS RAM). All components engage with the IDS Protocol component, depicted by dashed lines.
- **Remote Attestation:** Enhances trust among various components, verifying the integrity of software on other participants' systems (detailed in Section 4.1 of IDS RAM).

- **(Audit) Logging Service:** Logs comprehensive information during component operation, including system changes, errors, data access, and policy actions. It can forward data to systems responsible for auditable logging and provides or necessitates an interface to these systems.
- **Monitoring Service:** Monitors component status to verify conditions such as active operation, error presence, or offline status of the IDS Connector.
- **Data App Management:** Manages the downloading, deployment, and integration of IDS Apps within the IDS Connector.
- **Policy Engine:** Consolidates all elements enforcing IDS Usage Control Policies, including various points like PAP, PEP, PIP, PXP, PMP, and PDP.
- **Contract Management:** Oversees contract negotiations between Participants and archives the IDS Contract Agreements. Though it is an aspect of Metadata Management, its significance to Usage Control in IDS necessitates its distinction as a separate component.
- **Metadata Management:** Stores metadata of utilized or shared data assets, primarily governed by the IDS Information Model, but possibly supplemented with other details. It links contracts from the Contract Management component with data from the Data Management component.
- **Data Management:** Contains the data assets or references to data sources, destinations, or IDS Apps, enabling dynamic data retrieval or transmission.
- **Configuration Management:** Houses configuration parameters applicable to IDS Protocols and general components.
- **User Management:** Administers user authentication across all component interfaces, potentially utilizing external Identity Services or its own resources. It includes a configurable interface.

IDS Connectors vary in their technological implementations, each tailored to specific functional requirements. These connectors are classified by their certification level as stated in Section 4.2 of IDS RAM, signifying their adherence to certain security and data sovereignty standards.

5.4.2.2 Building blocks

5.4.2.2.1 IDS Connector

An IDS Connector is a component within the International Data Spaces (IDS) framework, facilitating data transactions among various entities. This component provides a gateway for data retrieval, preservation, and manipulation through Data Endpoints. Whether situated on-site or in cloud infrastructures, IDS Connectors employ container management technologies for applications, guaranteeing a secure, segregated space for IDS Apps and inherent Connector operations. Their implementation can vary, manifesting as Developer Connectors, Mobile Connectors, or Embedded Connectors, contingent on the operational context. The IDS Connector's primary roles include forging trustful relations, safeguarding data transfers, and endorsing uniform interoperability within the IDS network. Currently there are multiple implementations of IDS Connector (26 connectors, as of November 2023), that are listed in detail in the Data Connector Report [44] that is published (and updated on a monthly basis) by International Data Spaces Association. Among these connectors two options are being considered by the consortium. These are 1) Eclipse Dataspace Components[45] and 2) DataSpaceConnector[46] which are both available as open source.

[44] https://internationaldataspaces.org/data-connector-report/
[45] https://github.com/eclipse-edc
[46] https://github.com/International-Data-Spaces-Association/DataspaceConnector

5.4.2.2.2 Dataspace Protocol

The Dataspace Protocol [47]serves as a critical framework in the realm of data spaces, specifically designed to ensure technical interoperability among various participants. This comprehensive system mandates that any entity wishing to partake in a data space must adhere to the protocols outlined within this specification. The scope of these protocols extends beyond mere technical interoperability, encompassing aspects of semantic understanding, trustworthiness, organizational dynamics, and legal frameworks.

The protocol mainly functions as the tool to ensure:

- **Contextual Framework for Interoperability:** The protocol operates within data spaces, promoting a high degree of interoperability. While it lays the groundwork for technical communication and data exchange, it also encourages participants to engage in semantic interoperability, ensuring that data semantics are universally understood and consistent.
- **Multi-Level Interoperability:** Interoperability isn't just about systems working together. It's about building trust, establishing common organizational practices, and navigating the legal landscape. These elements are crucial for cross data space communication, which, while not covered in this document, are handled within the data spaces' organizational and legal structures.
- **Role of Connectors:** Participants interact through agents known as "Connectors." These are sophisticated entities that handle the protocols and enable data exchange, ensuring that interactions are seamless, secure, and adhere to the established guidelines. They may also interface with other systems as necessary, providing a versatile communication nexus within the data space.
- **Identity Management and Trust Framework:** A pivotal element within this ecosystem is the Identity Provider, tasked with authenticating participant agents and validating their claims. This mechanism can adapt to the unique structures of different data spaces, whether they're centralized, decentralized, or federated, underpinning the trust framework essential for secure and reliable interactions.
- **Beyond Technical Specifications:** Connectors may possess additional internal functionalities (e.g., monitoring, policy enforcement engines) that, while outside the purview of this specification, play a role in the broader operational context. Similarly, the protocol doesn't dictate the specifics of the data being transferred (like structure, syntax, or semantics), leaving that to the discretion of participant agreements, thereby ensuring flexibility and context-specific relevance.

In brief, the Dataspace Protocol is a foundational infrastructure that supports a multifaceted interoperability landscape, addressing not only the technical aspects but also the organizational, legal, and trust-based facets critical for robust, secure, and effective data spaces.

[47] https://docs.internationaldataspaces.org/ids-knowledgebase/v/dataspace-protocol/overview/readme

5.4.3 Process view

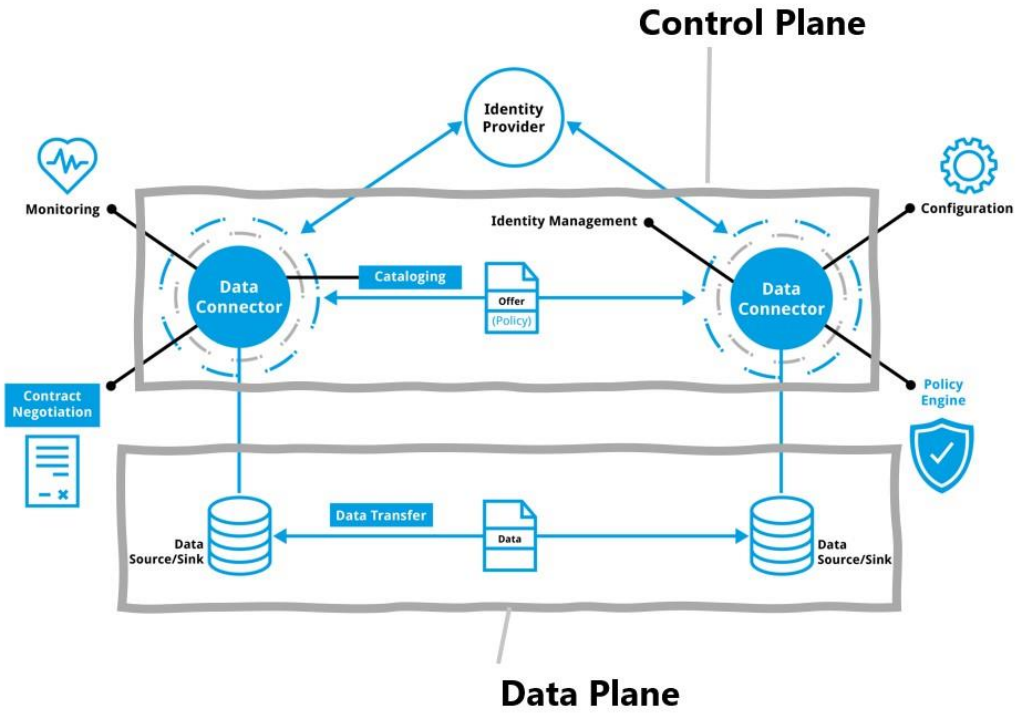


Figure 39: Separation of Control Plane and Data Plane in Dataspace Protocol (v0.8)

Figure 39 depicts the separation of Control plane and the Data plane in the Dataspace Protocol.

5.4.3.1 Sequence diagram

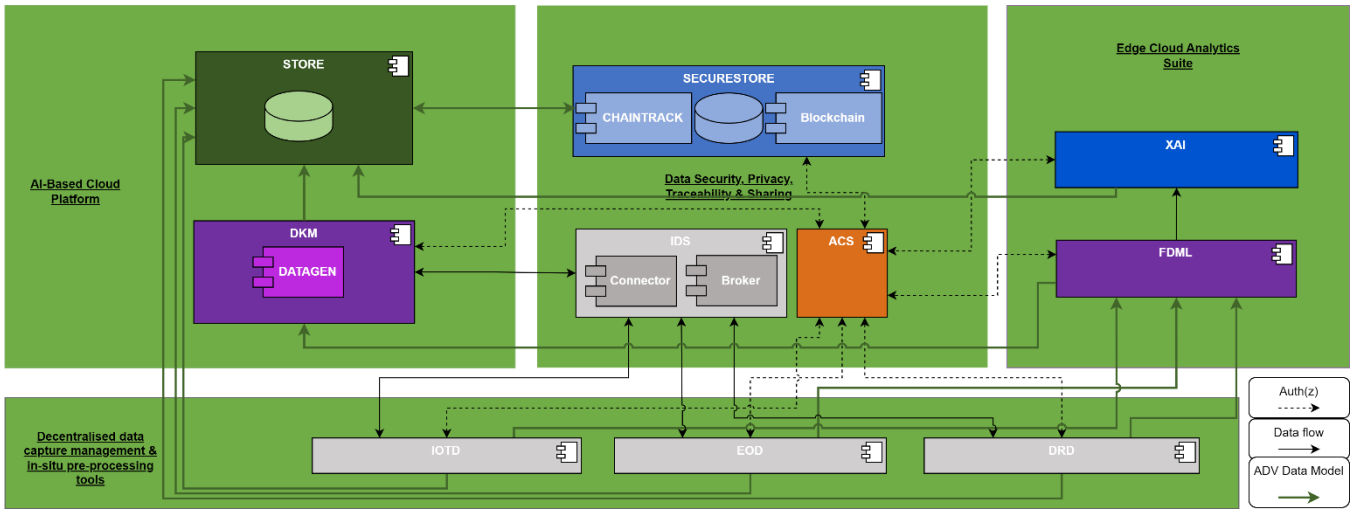


Figure 40. ADV platform functional view (source: deliverable D1.3)

In the current status of the ADV project, the implementation of IDS connectors within the ADV platform has not yet been realized. As seen in Figure 40, the IDS connectors are expected to interact with the majority of the ADV platform, meaning that the deployment of IDS connectors is contingent upon the completion and integration of all participating components within the ADV ecosystem. Consequently, the sequential process, accompanying diagrams, and detailed implementation specifics are not yet unavailable for description, but

will be included in the following iterations of the deliverables describing the technical implementations of the ADV platform.

5.4.4 Interfaces

The Interfaces, including the Data models and the APIs have not been defined yet. Details will be available in the next version of the deliverable.

5.4.4.1 Data models used in interfaces

N/A at the moment.

5.4.4.2 Description of APIs

N/A at the moment.

5.4.5 Technologies and implementation details

Implementation details are not available at the moment. Details will be available in the next version of the deliverable.

6 AI-Based Cloud Platform

6.1 Decentralised Knowledge Management (DKM)

6.1.1 Description

The main objective of the Decentralised Knowledge Management (DKM) component is twofold. On the one hand, it orchestrates the process of training and storing AI models in a federated manner, establishing communication among the FDML (see Section 4.1), the XAI (see Section 4.2), and the SECURESTORE (see Section 5.1) components. On the other hand, the DKM itself is responsible for generating the global AI model by aggregating the local or regional models from FDML, along with the associated metadata.

To achieve this, the DKM acts as the root server of the HFL topology described in Section 6.1. The basic functionality of this root server is to aggregate the local/regional model weights obtained and generate the metadata associated with the resulting global model. However, a fundamental aspect of the AgriDataValue platform is privacy and security, which, logically, must extend to the process of generating AI. For this reason, the FL performed in collaboration between the FDML and the DKM components, must include strategies to avoid data leakage and to strengthen the learning process against malicious attacks. Two solutions are proposed within the DKM for this purpose: an agent authentication process and poisoning attack detection.

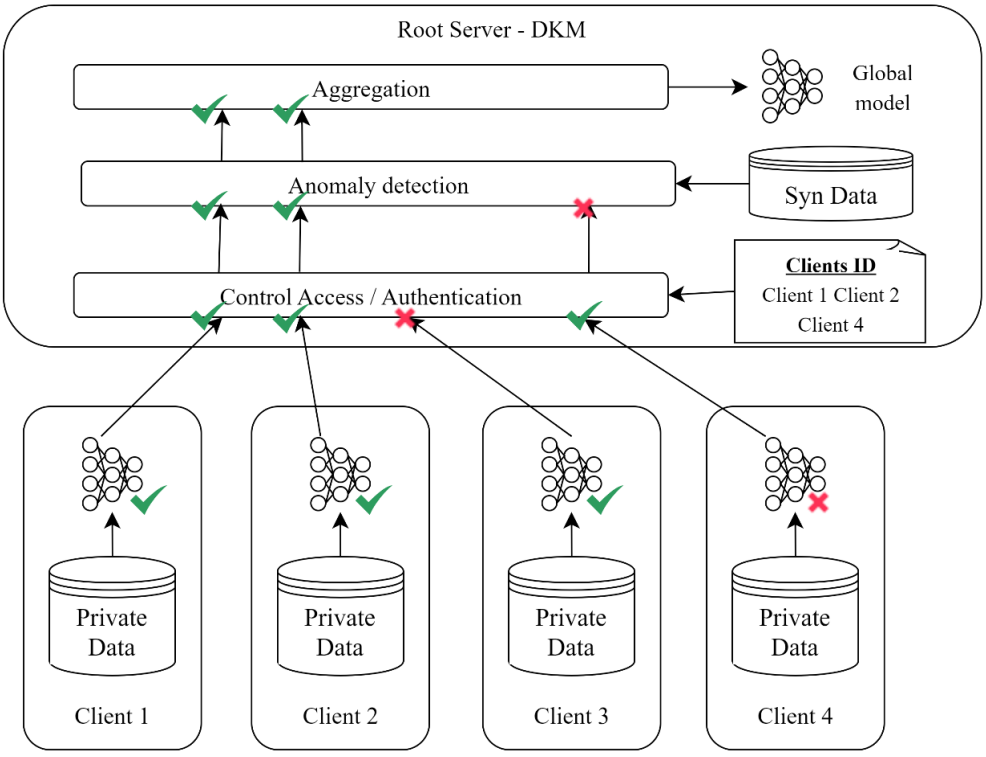


Figure 41: DKM security in the learning process

Regarding the authentication process, also called control access, the DKM verifies that the received weights originate from the address of one of the FDML agents (whether intermediate server or client) directly associated with the central server in the HFL topology. This process prevents unauthorized devices from participating in the FL process, thereby safeguarding the integrity of the final global model generated during the aggregation. This is the case of the third client in Figure 41 example.

On the other hand, a poisoning attack [14] refers to the malicious submission of corrupt model weights by a FL client. These corrupt weights significantly degrade the performance of the final global model. To address these attacks, the DKM in collaboration with the FL-AgriDataGen (section 6.3), proposes the combination of generative DL methods and anomaly detection algorithms to detect and discard poisonous weights for the aggregation. This process is based on the offline generation of synthetic databases that simulate real agronomical data by the FL-AgriDataGen component. Then, these synthetic databases are employed by the DKM to detect anomalous weights received from the clients and discard them. This is the case of Client 4 in Figure 41.

Finally, the aggregation is performed using the FedAvg technique. This technique is widely used in the FL state-of-the-art, proposing the direct aggregation of the model weights by the computation of their weighted average. However, throughout the technical development of the project, other alternatives may be investigated to potentially enhance the platform's final performance.

For the sake of simplicity, as can be seen in Figure 41 and Figure 42Figure 44, all the functionality of the DKM has been divided into three functional modules. These modules, their interactions, and their development are described in more detail in the subsequent sections.

6.1.2 Development view

6.1.2.1 Component diagram

Figure 7 depicts the component diagram for the DKM, illustrating its several sub-components and functional blocks. In a similar way to FDML, functional blocks are represented with dashed lines, sub-components with solid lines, and components outside the DKM are represented in gray.

It is important to note that, to align with the decentralized strategy proposed by the AgriDataValue platform, this component is exclusively engaged in the fulfillment flow. Hence, all the arrows between components represent interactions established during the process of learning and storing AI models tailored for different use cases.

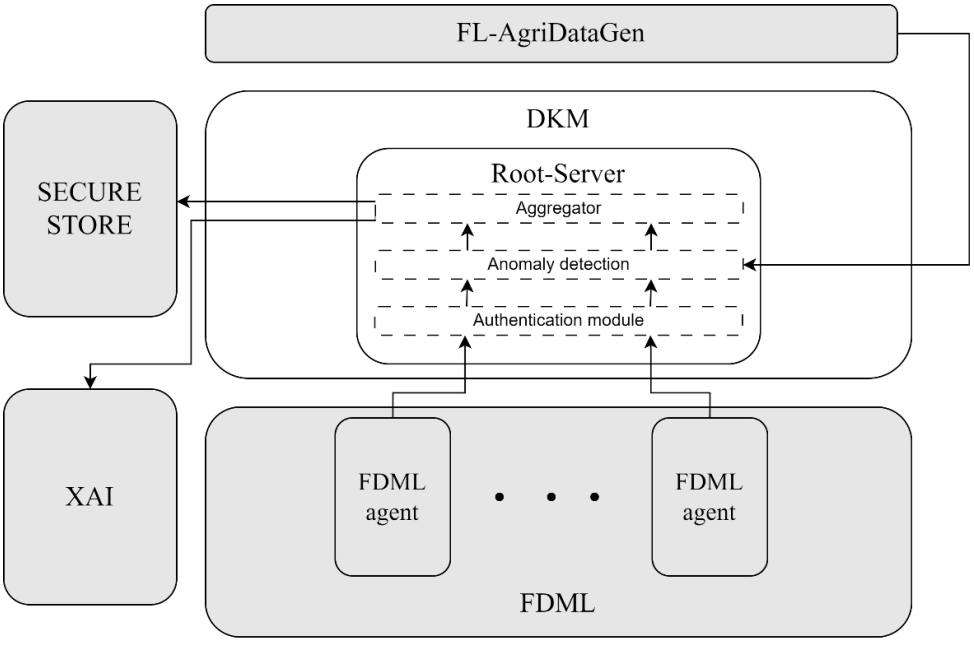


Figure 42: Logical diagram for the DKM component

6.1.2.2 Building blocks

As can be seen in the previous illustration, the DKM component directly interacts with the main components dedicated to the training and storage process of AI models, namely: the FDML component, the SECURESTORE, the XAI module, and finally the FL-AgriDataGen. As it was mentioned before, the functionality of DKM can be viewed as an extension of the traditional functionality of a root server in a HFL scheme. For the sake of clarity, this functionality has been divided into three functional blocks:

1. **Authentication module or control access**: This module receives the parameters/weights from local or regional models originating from the different FDML agents. Upon receiving these parameters, this module undergoes an authentication process where the origin address of the weights is verified. In those cases where the origin address is unknown, their respective weights are discarded from future analyses.
2. **Anomaly detection:** This module performs the anomaly detection process to detect and avoid poisoning attacks. To do this, this module evaluates the performance of the local models with synthetic data generated by the FL-AgriDataGen. Those models whose performance is below a specific threshold are excluded from the final aggregation process.
3. **Aggregator:** This functional block performs the aggregation process to conform to the global model and generates the metadata associated with the generated model. The employed aggregation technique is the FedAvg. Following this technique, the global model weights in a specific round of the hierarchical federated learning (w_t^G) are calculated as the weighted average of the weights of the FDML agents connected to it (w_t^k). The final equation is as follows:

$$w_t^G \leftarrow \sum^{K} \frac{n_k}{n} w_t^k$$

where n_k is the number of samples used for training local model of FDML client k. Additionally, this module generates the metadata associated with the global model. These metadata are going to be stored alongside the model in SECURESTORE. The next table contains a description of the final generated metadata. In addition to these metadata, the storage of information related to statistics of the training dataset will be studied.

Table 11: DKM - Metadata generated for each global model

Field	Format	Description	Example
Model ID	Unique id	Unique Id of the model. This ID is directly generated from the number of models available in the database.	1
Version	Integer	Version of the model	0
Date	Date	Date when the model has been generated	“15/12/2023”
Model type	String	Type of AI model trained. Initially AgriDataValue will focus on DL models such as DNNs, CNNs or RNNs among others.	“DNN”
Dependencies	List of strings	List of dependencies for running the model.	[“Tensorflow=2.14”, “numpy=1.26”]
Input features	List of strings	Name of the input features of the model.	[“Temperature”, “humidity”]
Feature types	List of strings	Data types required for the input variables.	[“float”, “float”]

Output features	List of strings	Name of the output features of the model.	[“Pressure”]
Output types	List of strings	Data types of the output variables.	[“int”]
Preproc	String	Type of pre-processing performed. Some examples are: “min-max norm”, “max norm” or “z-score”.	“max norm”
Data type	String	Type of the training data. Some examples are: “Tabular”, “Timeseries” or “images”	“Tabular”
Use case / Domain	String	Domain for which the data can be used. It may match the uses cases defined in ADV.	“Fertilization”
ADV Pilot	Int	ADV pilot number in which the data has been extracted.	8

6.1.3 Process view

6.1.3.1 Sequence diagram

The following diagram depicts the interaction of the DKM with the other components during the process of training and storing AI models. It is important to note that, for simplicity, the sub-components of FDML (namely FDML clients and FDML intermediate servers) have been grouped in this diagram under the FDML component. For more detailed information regarding the data flow among these sub-components, readers are referred to Section 4.1.

As seen in Figure 43, the sequence diagram can be divided into the following stages:

1. **Training launch:** The initialization of the training process is directly performed by the end-user of the platform through AgriDataValue’s front-end or via a POST request to the DKM service. Subsequently, DKM starts operating as the root server of the HFL process described in Section 4.1.
2. **Model initialization:** The DKM initializes the AI model and sends the initial model weights to the FDML.
3. **Global model generation:** Following the training process of the different local and regional models, the three functional modules are executed. Firstly, the authentication or control access of the FDML agents is performed. Then, the evaluation of the received model using the synthetic databases generated by the FL-AgriDataGen takes place through the anomaly detection module. Finally, the aggregation and the generation of the metadata is performed by the aggregator module.
4. **Global model evaluation:** The global model is forwarded to the FDML clients for on-device evaluation of the model with their local data.
5. **Global model storage:** Once the model has been aggregated, the DKM triggers the storage of the model in a pickle format along with metadata in JSON format.
6. **XAI trigger:** Following the model storage in the SECURESTORE, the DKM sends the associated model metadata to the XAI module. The XAI component, through its builder service, loads the previously stored model in SECURESTORE and trains an XAI explainer, which is also stored in SECURESTORE.

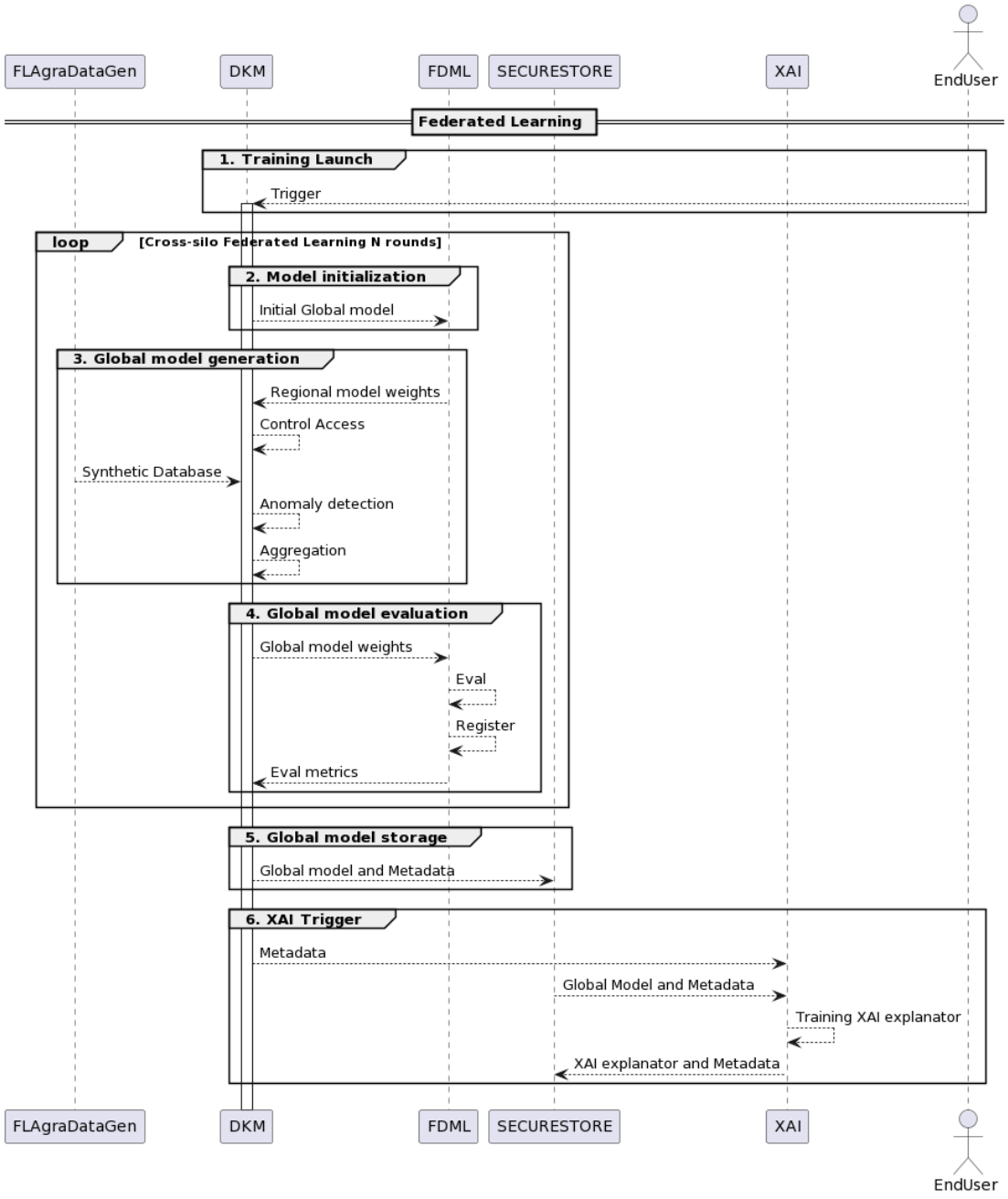


Figure 43: Sequence diagram for the DKM

6.1.4 Interfaces

6.1.4.1 Data models used in interfaces

Name	*DKM Data model*	
Property	**Type**	**Description**
Weights	Array of floats	Weights / parameters of the model trained
Metadata	JSON / Dict	Metadata (described in the table X) defined in a JSON format.

6.1.4.2 Description of APIs

(REST)

6.1.4.3

Title	*Training Launch*
URL: *This field holds the relative path to the described API. For simplicity Root path can be cut off from this description and can be placed as a hypertext above the API template*	
/rest/train	
Method *This field holds the type of the Method used*	
POST	
Data Params *This field holds the body payload of a post request.*	
Optional:	
Advusecase[str]	*Domain for which the data can be used. It may match the uses cases defined in ADV.*
Advpilot [int]	*ADV pilot number in which the data has been extracted.*
Success response <What should the status code be on success and is there any returned data? This is useful when people need to know what their callbacks should expect>	
200	*The global model has been trained successfully*
Error response *This field holds the list of all possible error responses. Doing that, helps prevent assumptions of why the endpoint fails and saves a lot of time during the integration process.*	
404	*The DKM is not running*
400	*Syntax error in the post command*
Sample call *This field holds a possible sample call to the described endpoint in a curl-like format. Please, choose the format wisely so that is clear and easy to read by the interested parties.*	

```
curl --location '0.0.0.0/rest/train' \

--header 'Content-Type: application/json' \
--data '{
   "advusecase":"Fertilization",
   "advpilot":8
}'
```

Title	*Model Aggregation*
URL: *This field holds the relative path to the described API. For simplicity Root path can be cut off from this description and can be placed as a hypertext above the API template*	
/rest/upstream	
Method *This field holds the type of the Method used*	
POST	
Data Params *This field holds the body payload of a post request.*	
Required:	
Weights[array of floats]	*Parameters of the local model.*
Optional:	
Metadata [json]	*Metadata specified in the format specified in table Y*
Success response	
200	*Local models correctly aggregated*
Error response *This field holds the list of all possible error responses. Doing that, helps prevent assumptions of why the endpoint fails and saves a lot of time during the integration process.*	
404	*The DKM is not running*
400	*Syntax error in the post command*
Sample call *This field holds a possible sample call to the described endpoint in a curl-like format. Please, choose the format wisely so that is clear and easy to read by the interested parties.*	

```
curl --location '0.0.0.0/rest/train' \
```

```
--header 'Content-Type: application/json' \
--data '{
    "weights":[[0.03,0.21,-0.03]],
    "metadata": {
        "modelID":1,
        "version":0,
        "date":"15/12/2023",
        "modeltype":"DNN",
        "dependencies": "["Tensorflow=2.14", "numpy=1.26"]",
        "inputfeatures": "["Temperature", "humidity"]",
        "featuretypes":"["float","float"]",
        "outputfeatures":"["Pressure"]",
        "outputtypes":"["float"]",
        "preproc":"max norm",
        "datatype":"Tabular",
        "advusecase":"Fertilization",
        "advpilot":8
    }
}'
```

6.1.5 Technologies and implementation details

The technology used for the implementation and deployment of the DKM is similar to the one described in Section 4.1.4 for the FDML. However, neither TensorFlow nor Pandas have been involved in the development as the DKM does not require training AI models or preprocessing data.

Annex A presents a detailed description of the development of the DKM with the Fleviden framework. This development has been Dockerized and deployed with the FDML with Docker Compose. The developments related to this component have been included in the first technical demonstration of the project, available on the project's GitLab repository[48]. It is important to remark that in the actual version of the component, the integration with the FL-AgriDataGen for the anomaly detection has not been performed yet.

6.2 Storage (STORE)

6.2.1 Description

The STORE component will address the needs for storing non-sensitive and non-private data generated from the rest of the components of the ADV platform. As of now, it is not yet clear the exact data that will be stored in this component. Nevertheless, the STORE component is planned to support several types of data storage needs, e.g. stream data, metadata, sensor data, etc. More thorough description of the component is planned for the next version of the deliverable.

6.2.2 Development view

6.2.2.1 Component diagram

The logical view of the STORE component is depicted in Figure 44. The component will expose a data management API which will handle all incoming data requests, supporting several communication protocols (e.g., HTTPS, TCP, MQTT), routing the data to the appropriate internal data store. Aiming to address the

[48] https://git.agridatavalue.eu/agridatavalue/demos/demo-fulfillment-flow

storage needs for several data types, it is planned to include a range of internal stores, i.e., SQL store, NoSQL store, in-memory store, etc. The data management layer will also handle any communication needed with the SECURESTORE. Of course, the needs of the data will modify the final sub-modules of the component.

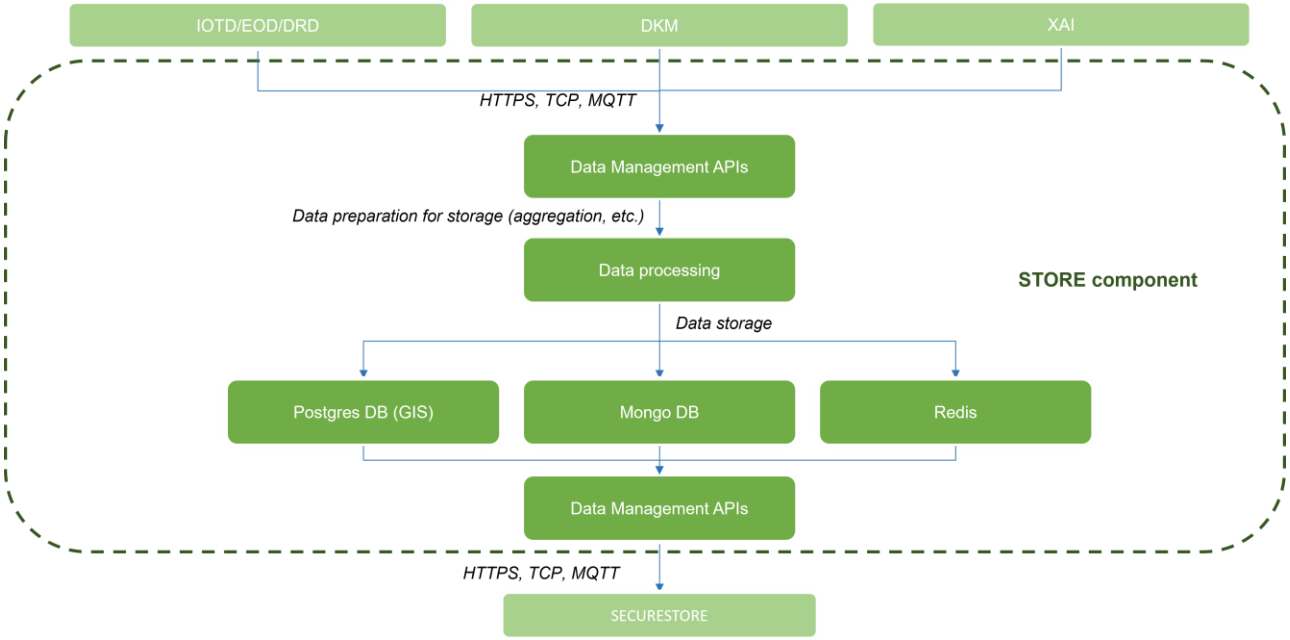


Figure 44: STORE - Component diagram

6.2.2.2 Building blocks

The building blocks of the component are the Data Management layer and the stores. However, these are not defined yet, hence, they will be described in the updated version of this deliverable.

6.2.3 Process view

6.2.3.1 Sequence diagram

The sequence diagram in Figure 45 shows the high-level steps for storing incoming data from the data sources to the internal stores of the STORE component, and the subsequent forwarding of (a part of) them to other components of the ADV platform (such as the SECURESTORE component).

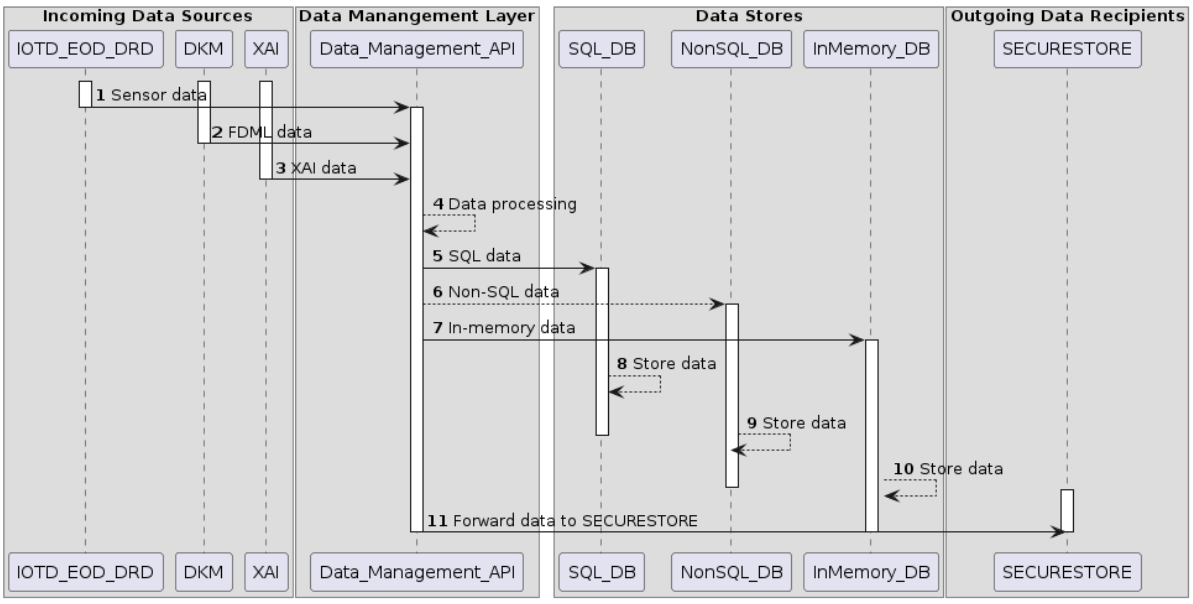


Figure 45: STORE - Sequence diagram

6.2.4 Interfaces

The Interfaces of the STORE component, including the data model and the API description of the Data management layer submodule are not yet defined; they will be described in the updated version of this deliverable.

6.2.4.1 Data models used in interfaces

The data model will follow the ADV data model specification which will be based on the AIM data model.

6.2.4.2 Description of APIs

The API of the STORE component is not defined yet.

6.2.5 Technologies and implementation details

As briefly mentioned, the STORE component will address the needs for several types of data. Therefore, it will leverage technologies that are able to handle this heterogeneity. As of now, the technologies that have been identified and are planned to be used are the following:

1. Postgres DB[49] including PostGIS[50] for the SQL DB store.
2. MongoDB[51] for the NoSQL DB store.
3. Redis[52] for the in-memory store.
4. Python + Django[53] for the Data Management layer API.

Additional technology and implementation details will be included in the next version of this deliverable.

6.3 FL-AgriDataGen (DATAGEN)

FL-AgriDataGen, short for Agri-environmental data generator, is a component that aims to the generation of synthetic labeled data samples, following ADV's systematic analysis. The component is going to leverage Generative Adversarial Networks (GANs), a state-of-the-art generative method. The purpose of the generated datasets will be two-fold: on the one hand, they will be employed as attack datasets to assist the evaluation and validation of the FDML anomaly detection algorithms. On the other hand, the datasets generated by the FL-AgriDataGen component will be employed for training purposes as and when needed, e.g. in the context of the XAI component. In the following paragraphs, an extensive description of the component and the employed technologies is presented.

6.3.1 Description

FL-AgriDataGen is going to utilise GANs for the generation of synthetic ADV datasets, to be used for both the evaluation of the FDML anomaly detection algorithms and, possibly, for the training of other components (e.g. XAI).

GANs have emerged as a powerful and innovative approach in the realm of dataset generation. Introduced by Ian Goodfellow and his colleagues in 2014 [15], GANs consist of two neural networks, a generator, and a discriminator, engaged in a competitive learning process. The generator creates synthetic data instances, attempting to mimic a given dataset, while the discriminator evaluates the authenticity of these generated

[49] https://www.postgresql.org/
[50] https://postgis.net/
[51] https://www.mongodb.com/
[52] https://redis.io/
[53] https://www.djangoproject.com/

samples in comparison to real ones. Through iterative training, the generator improves its ability to produce increasingly realistic data, while the discriminator refines its capacity to differentiate between real and synthetic instances. GANs have found applications in various domains, ranging from computer vision to natural language processing, enabling the generation of diverse and high-quality datasets for training machine learning models.

Our component will exploit the capabilities of GANs to generate synthetic data, which can subsequently be employed for tasks such as data augmentation, attack generation, and fortifying defense mechanisms. We construct a versatile GAN model capable of handling both image and tabular data, producing realistic samples aligned with the distribution of a given training set. To enhance accessibility, we encapsulate our model within an API, facilitating seamless integration with larger components for training, testing, or making inferences through endpoints. Additionally, we package our GAN model as a Docker container, thus streamlining deployment and installation procedures across diverse locations. Currently, we have launched the development of an image GAN tailored to olive tree images. As soon as pilots' historical data become available, the GAN model is going to be expanded to incorporate additional model and dataset options (e.g. tabular), in order to support the multi-modal data source approach of ADV, i.e. data collected via the IOTD, EOD and DRD components.

6.3.2 Development view

6.3.2.1 Component diagram

In this paragraph, the sub-components of FL-AgriDataGen and their intercommunication and operational flow are presented. Two similar data flows are implemented in the context of the component. In any case, the end-user (or component) must provide the training dataset, based on ADV's real data. Subsequently, the training, inference or other procedures are triggered either manually or via the available API.

The input dataset is processed in an appropriate manner by the DATAGEN pre-processor component, to be later fed as input to the GAN model. The GAN model processes the input dataset to generate the ADV synthetic data, as needed according to type of input dataset. The output of the GAN model passes on to the ADV Data Model Translator to be translated in the defined ADV data model format. The results generated by the GAN model, and translated to the ADV data model are then retrievable via the API, thus providing a streamlined and responsive interaction between external components and the underlying model. The generated ADV synthetic dataset will then be given as input to the FDML component for the validation of the developed anomaly detection algorithms. Additionally, the generated synthetic data will also be provided for training purposes to other ADV components (e.g. XAI), if / when needed.

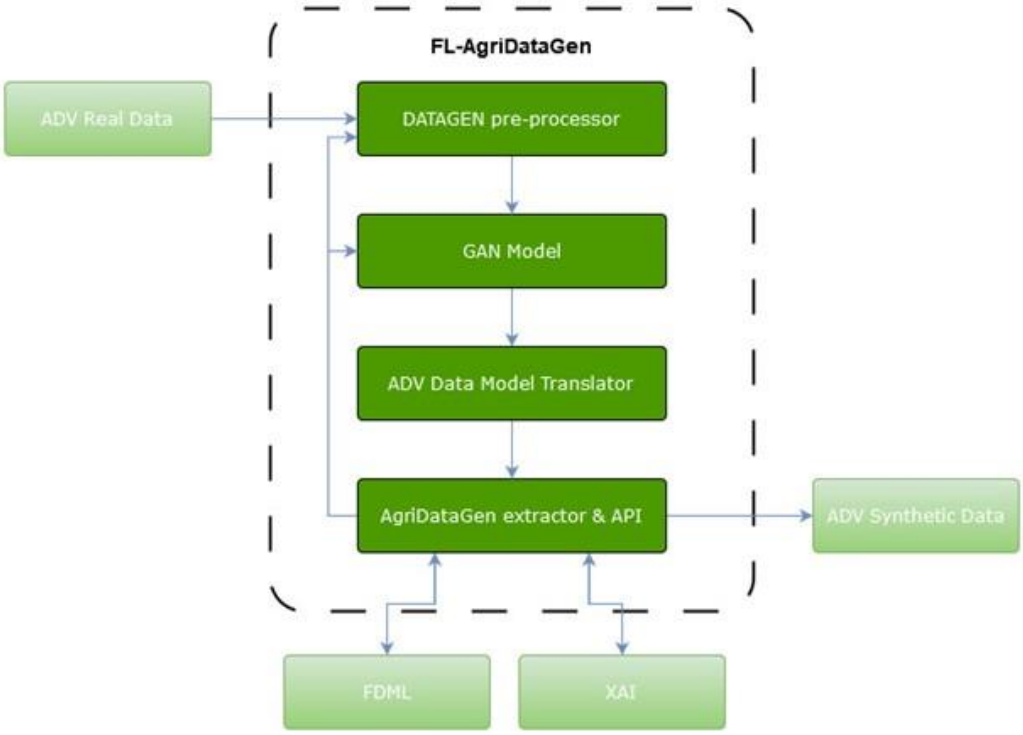


Figure 46: FL-AgriDataGen component diagram

6.3.2.2 Building blocks

6.3.2.2.1 DATAGEN Pre-processor

The **DATAGEN pre-processor** is the sub-component that is responsible for the pre-processing of the ADV real data that are given as input to the FL-AgriDataGen. Appropriate preprocessing procedures are followed in the case of both image and tabular input data.

As far as the input images are concerned, initially, they are resized to fit the input size of the GAN model. Resizing is an essential preprocessing step that adjusts the dimensions of an image to a desired size, facilitating consistent input dimensions for machine learning models and computational efficiency. Properly resized images ensure that the model receives uniform input sizes, which is crucial for achieving consistent performance and accurate predictions across diverse datasets.

The next step is normalization which is a crucial preprocessing step that scales pixel values to a consistent range, typically between 0 and 1, ensuring uniform brightness and contrast across images. By standardizing the intensity levels, normalization aids in reducing the variability in image data, making it more suitable for machine learning algorithms that are sensitive to input feature scales. This process helps in improving the convergence speed and overall performance of models trained on the normalized image data. We should underline that an appropriate normalization is also applied in the case of tabular data.

6.3.2.2.2 GAN Model

The **GAN Model** component comprises a collection of functions responsible for initializing and interacting with a GAN model. These functions are invoked by the API in response to specific HTTP requests, or manually via an appropriate command given via the command-line. Among these functions is a training function that initializes and trains a GAN model using the provided dataset. Additionally, there is an inference function that loads the model and generates appropriate outputs, along with various utility functions that contribute to the GAN training or inference processes. The overall approach follows the common practice in neural networks with respect to training, testing, and validation.

The model architecture employed for the image generation is a GAN. This model operates on a unique principle of competition between two neural networks: the generator and the discriminator. The generator is tasked with creating synthetic data from random noise inputs. Concurrently, the discriminator's role is to distinguish between these generated images and real ones from a dataset. As training progresses, the generator refines its output to produce images or tabular data that become increasingly indistinguishable from real ones, while the discriminator becomes more adept at making accurate distinctions. This adversarial process results in the generation of highly realistic data, with the interplay between the two networks driving improvements in both data quality and the generator's ability to produce authentic-looking content.

6.3.2.2.3 ADV Data Model Translator

The **ADV Data Model Translator (DMT)** is going to be responsible for the translation of the synthetic data to an ADV representation of them. This component will be based on the ADV data model translation library that will be created in the context of IOTD toolbox. The output of this sub-component will be a generated synthetic dataset that will be represented in the exact manner ADV's real data are represented.

6.3.2.2.4 AgriDataGen extractor and API

The **AgriDataGen extractor and API** component serves as a gateway for handling incoming requests seeking interaction with the GAN model through its designated endpoints, as well as an extractor and package manager for the datasets that are requested for generation. To be precise, the API exposes three endpoints catering to both training and inference functionalities. The training endpoint necessitates no parameters but mandates users to mount the input dataset to a specific location of the running container. Subsequently, the model undergoes training via this request, employing the adversarial GAN training process, and the generated synthetic datasets are then stored for future utilization, after their translation to the ADV Data model. On the other hand, the inference endpoint requires a size parameter, specifying the desired quantity of generated samples (e.g. images). Leveraging the previously trained model, this endpoint invokes the model to produce the requested number of data samples.

6.3.3 Process view

6.3.3.1 Sequence diagram

The sequence diagram is composed of two sequences, i.e. the training and inference sequence. The training sequence begins with an external ADV component issuing a training request to the AgriDataGen Extractor and API component, for it to be forwarded to the DATAGEN pre-processor component, for the appropriate pre-processing of the given dataset to be conducted. The pre-processed dataset is then given as input to the GAN model, and the actual training procedure follows. Upon completion, the training status is provided to the AgriDataGen Extractor and API, to be later accessed by the component which originally requested the training.

The inference sequence begins with an inference request by an external ADV component. Of course, the training should have been completed before the inference. The number of data samples to be generated is a required parameter of this step. The GAN model proceeds to the generation of the synthetic dataset, which is then translated by the ADV Data Model Translator. The ADV synthetic data are then provided to the requesting entity to be utilised as / when needed.

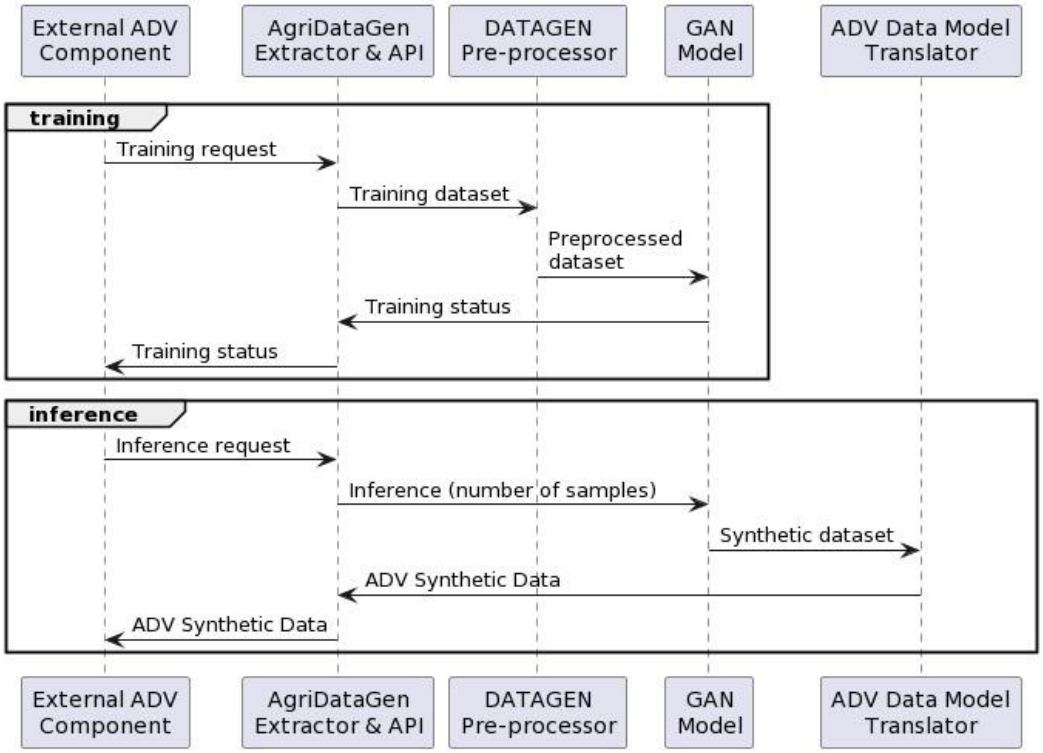


Figure 47: FL-AgriDataGen sequence diagram

6.3.4 Interfaces

6.3.4.1 Data models used in interfaces

FL-AgriDataGen will adopt the ADV data model. The synthetic data generated in this component will be translated before their extraction and provision to other ADV components. As the ADV Data model is not considered stable in its current version, the data model mappings will not be included in the present document. However, we should notice that the approach for the FL-AgriDataGen component will be, for the most part, identical to the approach adopted in the context of IOTD component.

6.3.4.2 Description of APIs

An initial version of the API provided by the FL-AgriDataGen component is recorded here. In this initial version, the API includes only three endpoints, two for training purposes and another one for inference. As the development of FL-AgriDataGen proceeds, additional essential endpoints will be created accordingly. We should also notice that in case of training, the input datasets must be provided in a specified format which will be finalised soon, thus it is not recorded in the present deliverable.

Title	*Training*
URL	
/api/v1/training/	
Method	
POST	
URL Params	
N/A	
Request Body	

<pre>{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "type": "object",
  "properties": {</pre>	

<table>
<tr><td colspan="2"><pre>
    "dataset_path": {
      "type": "string",
      "description": "The path of the training dataset.",
    },
    "dataset_type": {
      "type": "string",
      "description": "The dataset type",
      "enum": ["image", "tabular"]
    }
  },
  "required": [
    "dataset_path",
    "dataset_type"
  ]
}
</pre></td></tr>
<tr><td colspan="2">Headers</td></tr>
<tr><td>Authorization</td><td>Bearer <Token></td></tr>
<tr><td>Accept</td><td>application/json</td></tr>
<tr><td colspan="2">Success response</td></tr>
<tr><td><pre>
202

{
  "$schema": "http://json-
schema.org/draft-04/schema#",
  "type": "object",
  "properties": {
    "id": {
      "type": "integer",
      "description": "The ID of the
training.",
    }
  },
  "required": [
    "id"
  ]
}
</pre></td><td>Accepted training request</td></tr>
<tr><td colspan="2">Error response</td></tr>
<tr><td>400</td><td>Bad Request</td></tr>
<tr><td>401</td><td>Unauthorized</td></tr>
<tr><td>403</td><td>Forbidden</td></tr>
<tr><td>500</td><td>Internal Server Error</td></tr>
</table>

Title	*Training Status*
URL	
/api/v1/training/{training_id}/status/	
Method	
GET	
URL Params	
training_id=[integer]	The ID of the training to request the status for.
Request Body	
N/A	
Headers	
Authorization	Bearer <Token>
Accept	application/json
Success response	

200
--- 	
{
"$schema": "http://json-schema.org/draft-04/schema#",
"type": "object",
"properties": {
"status": {
"type": "string",
"description": "The status of the training procedure.",
"enum": ["failed", "pending", "running", "completed"]
}
},
"required": [
"status"
]
}
	OK

Error response	
401	*Unauthorized*
403	*Forbidden*
404	*Not found*
500	*Internal Server Error*

Title	*Inference*
URL	
/api/v1/inference/	
Method	
POST	
URL Params	
N/A	
Request Body	

```
{
  "$schema": "http://json-schema.org/draft-04/schema#",
  "type": "object",
  "properties": {
    "training_id": {
      "type": "integer",
      "description": "The ID of a completed training procedure."
    },
    "size": {
      "type": "integer",
      "description": "The number of data samples to generate. "
    }
  },
  "required": [
    "training_id",
    "size"
  ]
}
```

Headers	
Authorization	Bearer <Token>
Accept	application/json
Success response	
202	*Accepted inference request*

--- 	
{
"$schema": "http://json-schema.org/draft-04/schema#",
"type": "object",
"properties": {
"id": {
"type": "integer",
"description": "The ID of the inference.",
},
"dataset_path": {
"type": "string",
"description": "The path of the generated dataset.",
}
},
"required": [
"id",
"dataset_path"
]
}	

Error response	
400	*Bad Request*
401	*Unauthorized*
403	*Forbidden*
404	*Training ID not found*
500	*Internal Server Error*

6.3.5 Technologies and implementation details

The Model sub-component has been developed using Python, employing common libraries such as NumPy[54] for efficient computational operations. PyTorch[55] and Torchvision[56] were chosen as the framework for creating the model and conducting pre-processing steps. The API, responsible for handling interactions with the model, is built utilising Flask[57].

To ensure portability and ease of deployment, the system is containerized using Docker[58]. This encapsulation facilitates consistent performance across different environments and simplifies the overall deployment procedures.

[54] https://numpy.org/
[55] https://pytorch.org/
[56] https://pytorch.org/vision/stable/index.html
[57] https://flask.palletsprojects.com/en/3.0.x/
[58] https://www.docker.com/

7 Infrastructure and tools

AgriDataValue employs the DevSecOps methodology for development, integration, testing, and deployment. This section outlines and discusses the integration framework and tools that AgriDataValue used to build the project's CI/CD pipeline.

AgriDataValue adopts DevSecOps practices to support the platform's integration efforts and the management of the upcoming AgriDataValue platform releases. DevSecOps is, to put it simply, the expansion of DevOps security enhancements. DevOps with is a set of practices that combines software development and information technology (IT) operations. It is designed to be used in conjunction with Agile software development. Its ultimate objective is to accelerate the systems development life cycle while consistently delivering high-quality software. Because DevSecOps provides security by design, infrastructure security is integrated from the beginning into DevOps.

The phases of the AgriDataValue DevSecOps approach, given best practices, consist of:

- Arrange and produce. Procedures that have a direct bearing on the methods used in software design and development.
- Check, wrap, and distribute. These phases have a direct bearing on automated CI/CD tools, particularly GitLab, which handle continuous integration and delivery.
- Set up, identify, react, anticipate, and modify. These phases primarily relate to production-level quality assurance testing and the procedures involved in relaying the results back to the design and development phases in a continuous, looping exchange.

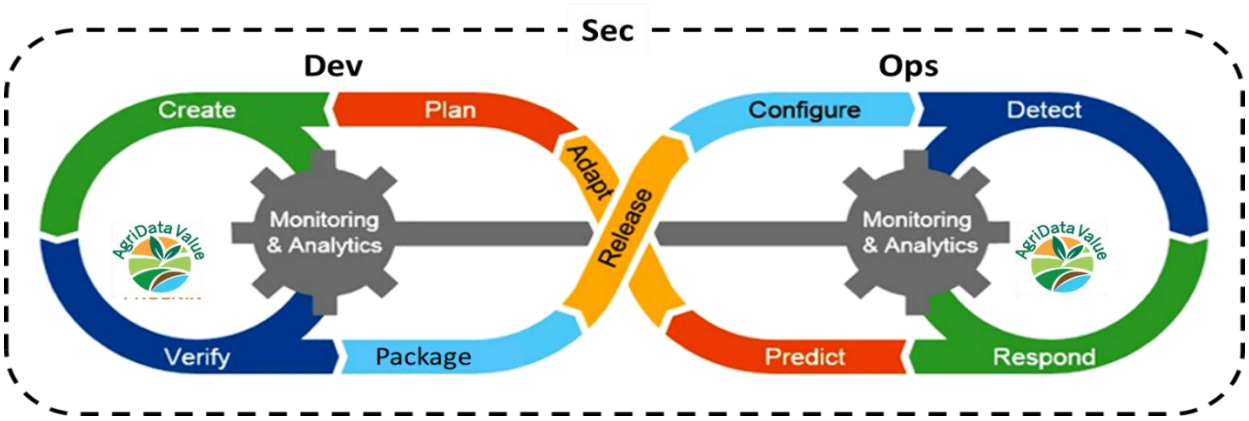


Figure 48: AgriDataValue DevSecOps

DevSecOps, or Development, Security, and Operations, is the process of automating the integration of security at every stage of the software development lifecycle, from initial design to integration, testing, deployment, and software delivery. It implies that security is an essential part of agile DevOps approaches and methodologies, such as continuous delivery, continuous integration, and cooperation.

7.1 CI/CD pipeline

Continuous Integration (CI) is a development methodology that uses a lot of automated tests, the appropriate tools to support automation, and incremental changes to maintain a working system through frequent (often daily) mainline integration. This makes it possible for teams to work together on shared code, which enhances system quality and development visibility. Continuous Integration (CI) is a development methodology that

typically calls for the application of Test-Driven Development (TDD) in conjunction with continuous refactoring. When a developer drives and unit-tests their local copy of the code, they ensure that it operates consistently.

Continuous deployment, or CD, is the process of automatically introducing new system releases into the live environment. When a system reaches a maturity level (as indicated by specific, predefined criteria), CD takes care of automatically updating an already-running version of the system to minimize downtime after the above-described continuous integration process.

When combined, CI and CD form a pipeline that takes in new developments and outputs an updated, functional system that is hosted in a predefined environment. In AgriDataValue, a CI/CD environment with GitLab has already been established. Figure 49 displays the high-level CI/CD pipeline of GitLab.

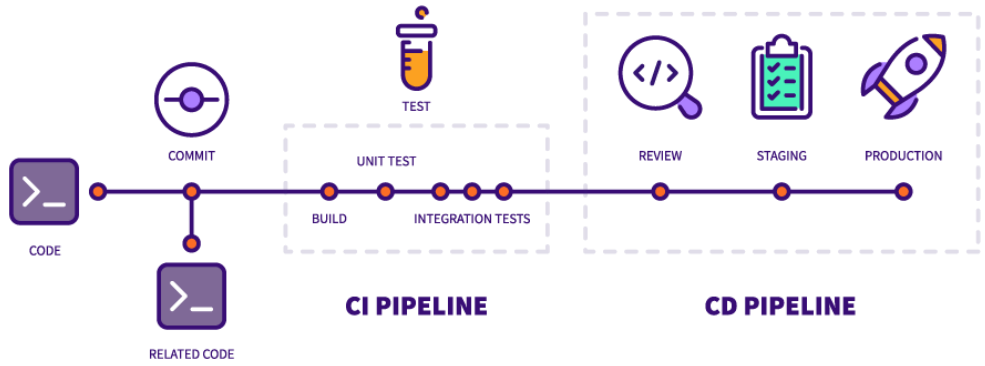


Figure 49: CI/CD pipeline – Source: about.gitlab.com.

The following figure provides a more detailed illustration of the suggested CI/CD pipeline.

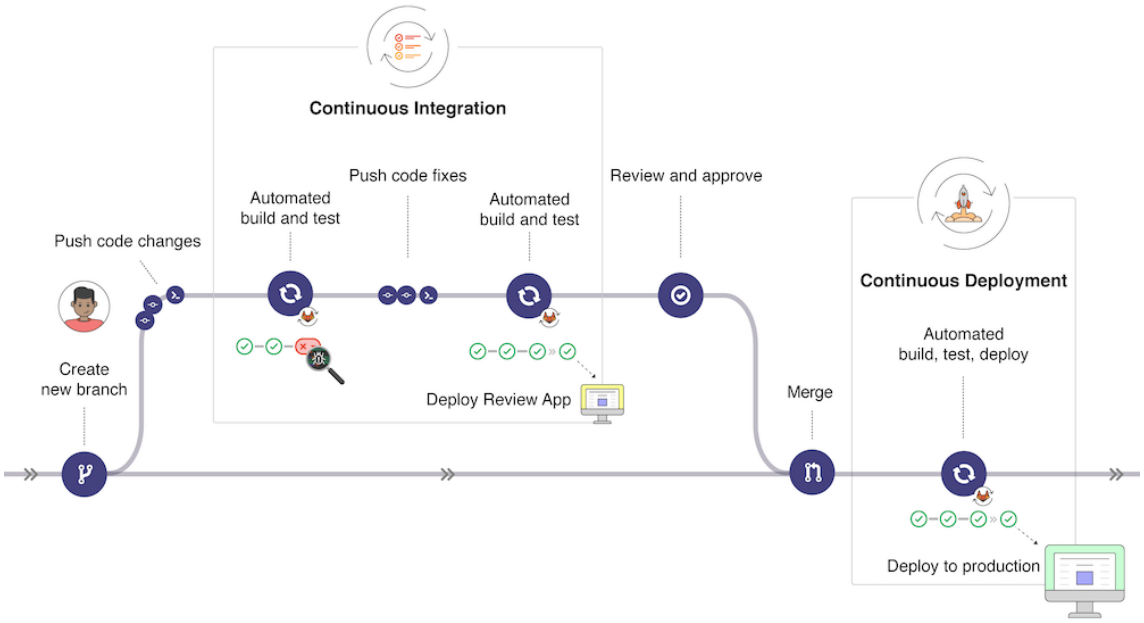


Figure 50: CI/CD pipeline steps - Source: docs.gitlab.

Figure 50 illustrates the primary steps of GitLab's CI/CD process in more detail This process can be explained as follows.

- When integrating a particular piece of software, the software developer aims to integrate it with a module's functionality.

- Unit-tests are presented by the developer for the relevant code segment, evaluating the output consistency of the module.
- Using a remote GitLab repository branch that the CI infrastructure has designated as a development branch, the developer pushes and commits changes to the code in their local repository.
- The developer has asked for code merging.
- The chosen unit test or tests are executed by the CI platform, and the project-specific CI/CD pipeline is turned on.
- If unit-testing is successful, additional integrated system testing is applied to the code.
- The recently committed source code will be incorporated into the code's master branch upon approval of the merge request, provided that the integrated system testing is successful.
- If not, the request will be denied. In this case, the developer must either update the code to follow the protocols or update the unit testing itself.
- The newly written code is now running on the deployment infrastructure and undergoing user testing and production following a successful CD, which is the process of building and deploying the software package via the source code management platform.

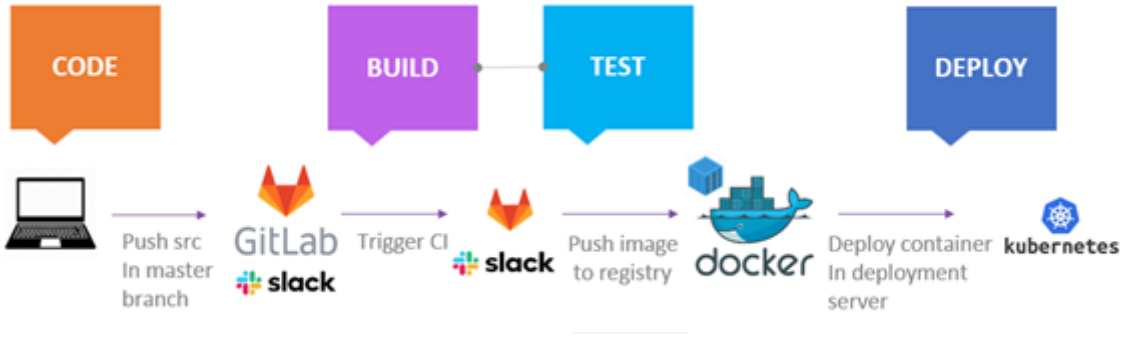


Figure 51: Development lifecycle (from source code to Kubernetes)

7.2 Tools in AgriDataValue

Within the context of AgriDataValue's integrated framework, the project's private infrastructure and toolbox have been set up and configured to support project operations, such as code sharing and development, testing, issue reporting, and user documentation. The sections below provide descriptions of the tools and infrastructure that have been used in AgriDataValue thus far.

7.2.1 Software management tool

It has been determined to store project developments and use source code management and version control through a private Gitlab deployment in AgriDataValue. The selection of GitLab (self-managed) was influenced by the following factors:

1. It is a Git-based open-source code management system.
2. Multiple projects, each with a unique set of groups and subgroups, may be created. This makes it possible to arrange components and source code in accordance with additional guidelines provided by the partners or with the architectural blocks that they are a part of.

3. Private actions can be taken during the early stages of software development while more stable versions are released as open source, as multiple private projects can be started.
4. The branching, developing, testing, and reviewing features of GitLab enable development teams to complete tasks in parallel before combining them.
5. Pipelines for automated deployment and integration procedures are provided by GitLab. Sequential continuous delivery (CD) and continuous integration (CI) operations are described by pipelines.
6. The ability to designate a project as private or public allows for the possible public release of source code and individual components.
7. Container images uploaded to Gitlab's private Container Registry can be used by the Kubernetes deployment.

As a result, the project's private GitLab instance, which requires admin permission and registration to access, is presently hosting the project developments.

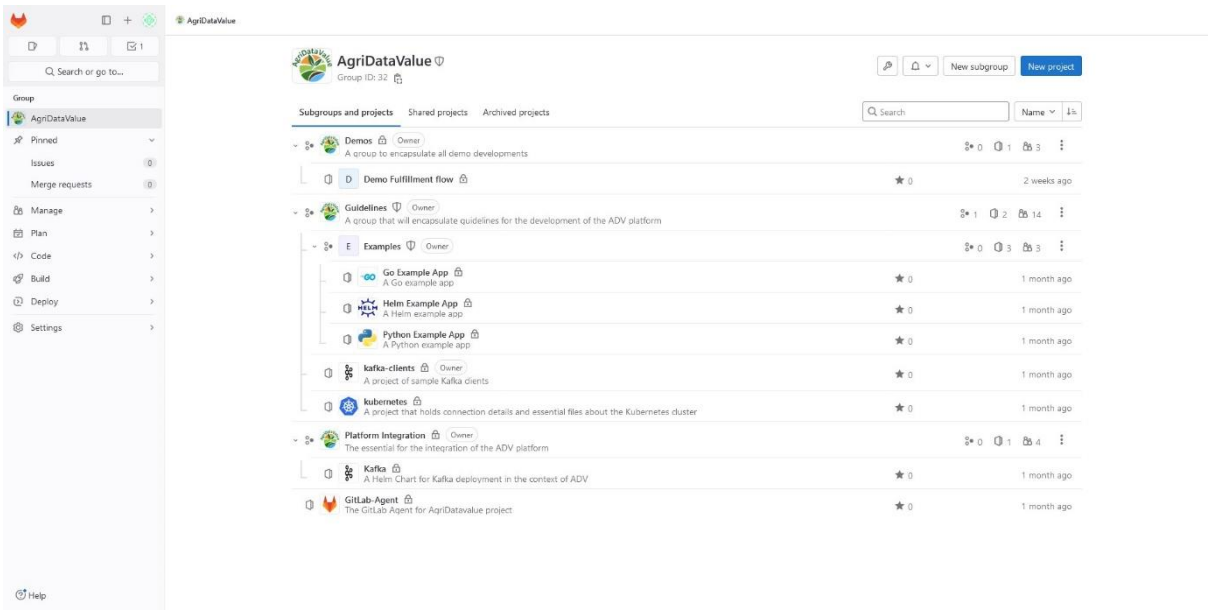


Figure 52: Overview of AgriDataValue group in ADV's self-hosted GitLab instance

7.2.1.1 Source code management

Tracking modifications to the source code is the primary objective of version control systems (VCS), also known as source code management (SCM). SCM is considered to be a crucial phase in the creation of software. Branching and merging, traceability, and the entire long-term change history of every file are the key features of this kind of system. Programmers, developers, and testers can make sure they are always working with accurate, up-to-date code, and avoid conflicts when combining code from different sources by maintaining a running history of changes made to a codebase. Git is a distributed software that is free and open-source, and it is the most widely used VCS tool. The AgriDataValue consortium uses GitLab, an open-source Git management tool, to manage Git repositories.

7.2.1.2 Issue tracking

An issue tracking system is a piece of software that makes it easier for members of a working group to create, manage, and discuss issues from the point of origin to the point of resolution. "Issues" in software can include any kind of software bug, desired new feature, unexpected behaviour, enhancement, or request for software to be updated, changed, added, or even for application functionality. Project planning is made easier by a number of features that the Issue Tracking System offers in addition to general issue management and

tracking. These consist of the capacity to allocate resources, establish deadlines, prioritize tasks, interact with other members of the team, send email alerts, and more.

Throughout the project, the AgriDataValue consortium will utilize the issue tracking tools offered by the GitLab web platform. There is a dedicated issue tracker for the AgriDataValue Integrated Platform. Issues, bugs, and suggestions for new features can be reported by AgriDataValue Integrated Platform users (developers, other stakeholders, and end users). The AgriDataValue team looks over these reports and takes the best possible action. The figures below show the issue tracker created for testing in two different views: the Board view, which displays all issues, and the List view, which displays the five "buckets" (referred to as lists in GitLab terminology): "Backlog," "ToDo," "InProgress," "UnderReview," and "Done." It is available and it remains at the disposal of the component owners and the end-users for use.

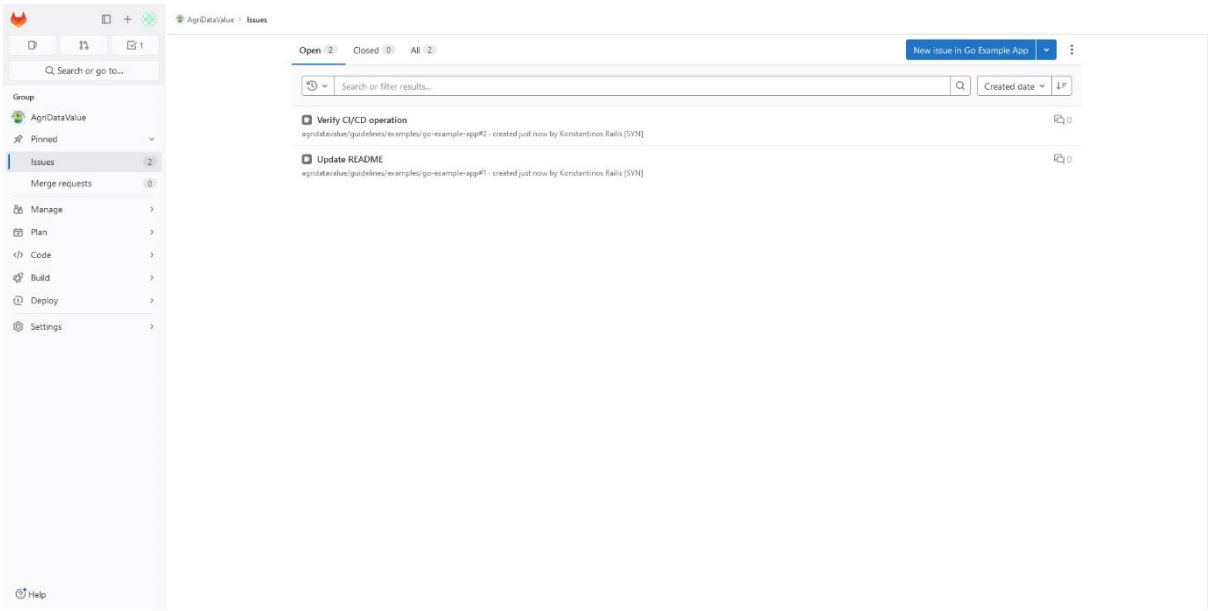


Figure 53: Snapshot of ADV's group issues in GitLab

7.2.1.3 Containerisation tools

Operating system (OS)-level virtualization, which is used to distribute and execute distributed applications, is known as containerization. Containerization, a lightweight alternative to virtual machines, entails enclosing an application within a container that has its own operating system. Code and library application-layer dependencies are packaged, and containerization builds an abstraction layer on top of it. Because they virtualize the operating system rather than the hardware, containers are dependable and portable in a variety of computing environments. Moreover, multiple containers can use the same OS kernel. Containerized software is compatible with both Windows and Linux applications and operates in an identical manner regardless of the infrastructure. With the aid of containers, software can be isolated from its surroundings and guarantee dependable operation even in the event of variations, such as between development and staging environments.

Docker is the most widely used application of containerization technology. An executable software package that is small and standalone is called a Docker container image. Code, runtime, system libraries, system tools, and settings are all contained in it. The most popular automated platform for container deployment, scaling, and management that is compatible with Docker containers is Kubernetes (K8s), an open-source system for automating the deployment, scaling, and management of containerized applications.

The technical developers of the project will put their completed technical outcomes into Docker containers and upload them to a registry repository that has the image changes. The AgriDataValue GitLab container

registry, accessible at https://registry.git.agridatavalue.eu , is utilized by the AgriDataValue project. The figure below shows a snapshot of this repository.

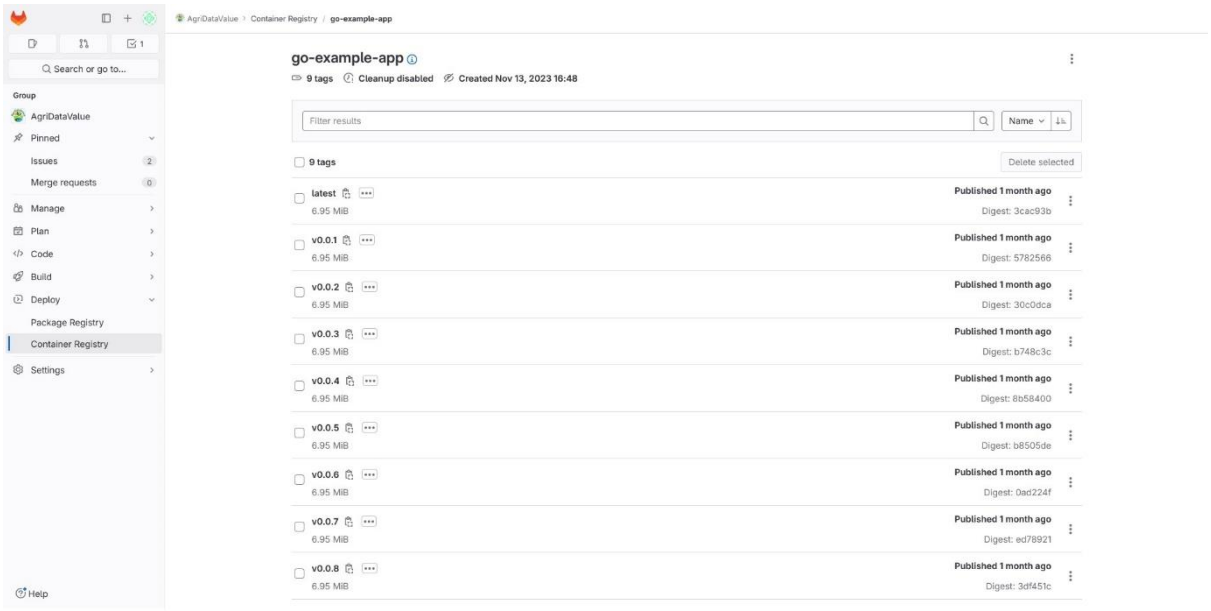


Figure 54: Snapshot of ADV's container registry

7.2.2 Message bus

As the technical partners discussed how to integrate the various components and exchange information, it became clear that a message bus was necessary. This tool would serve as a message broker for event messages sent between components. These messages may also be meant to function as alerts or event notifications, as opposed to being data messages.

The AgriDataValue message bus is provided by Apache Kafka, which can also be used with pub/sub paradigm data streams. It has been adjusted, configured, deployed, and tested under WP2 and Task 2.4 guidelines. It can be accessed by AgriDataValue components. Apache Kafka is a high-performance (low latency and high throughput) distributed streaming platform for managing real-time data. Large volumes of data can be handled and ingested into processing pipelines for batch and real-time applications with ease.

The main characteristics and functions of the platform are as follows: a) High fault-tolerance - Resistance to node failures and support of automatic recovery, b) Elasticity - High scalability, c) Distributed messaging system, d) Interoperability with all popular data sources and modern data storage technologies, e) Security (authentication, authorization, and encryption).

AgriDataValue's message bus is not visible to the public and is used to serve components that are either part of the same private network or the same Kubernetes deployment; Kafka can be used in already included in the deployment. Annex I contains a description of the Kafka messages' data schema.

7.3 Infrastructure

Because of the nature of AgriDataValue developments and the sensitivity of the data that may be involved in the overall project integration and component evolution (e.g. regarding model training), two dedicated physical servers have been used to host the integration platform. The technical specifications of the two physical servers are summarized in Table 12 and Table 13.

Table 12: Integration server overview: DELL PowerEdge R540

Manufacturer	DELL
Model	PowerEdge R540
No CPUs	1
CPU Type	Intel(R) Xeon(R) Silver 4210R CPU @ 2.40GHz
No Cores	10
No Logical Threads	20
RAM	96GB
Disk	3.2TB, RAID-5

Table 13: Integration server overview: DELL PowerEdge R210II

Manufacturer	DELL
Model	PowerEdge R210II
No CPUs	1
CPU Type	Intel(R) Xeon(R) CPU E3-1230 V2 @ 3.30GHz
No Cores	4
No Logical Threads	8
RAM	16GB
Disk	1TB, RAID-1

The PowerEdge R540 server functions as an integration server, hosting the AgriDataValue components and enabling their operation, while the PowerEdge R210II is mainly utilized for supporting services, such as hosting the project's Gitlab instance and the runners that go along with it. There is only one configured Gitlab runner at the moment, but more can be added as needed for the project's parallel builds.

In addition to an intrusion detection system (Suricata instance), a reverse proxy and reverse DNS server (HAProxy instance), and a pfSense firewall, the two servers previously mentioned are protected by the HAProxy service, which exposes all of the services provided by the AgriDataValue integration platform and also handles TLS termination (SSL offloading) for HTTP(s) calls made against the AgriDataValue component services.

7.4 Documentation

Wiki pages, word documents, and readme files have been made to help AgriDataValue technical partners integrate their components with the AgriDataValue tools (Keycloak, Kafka). Developers and integrators are the intended audience for this internal documentation, which also includes usage examples and details on the various tools, policies, and deployment procedures. Figure 55 shows an example of the contents of the Guidelines on GitLab and Figure 56 an example of a README file. In addition to integration/technical

information, wiki pages could be developed with the goal of instructing end users on how to use and administer tools and components. Both tool administrators (such as Keycloak) and will find use for this information as documentation. This documentation is currently in progress.

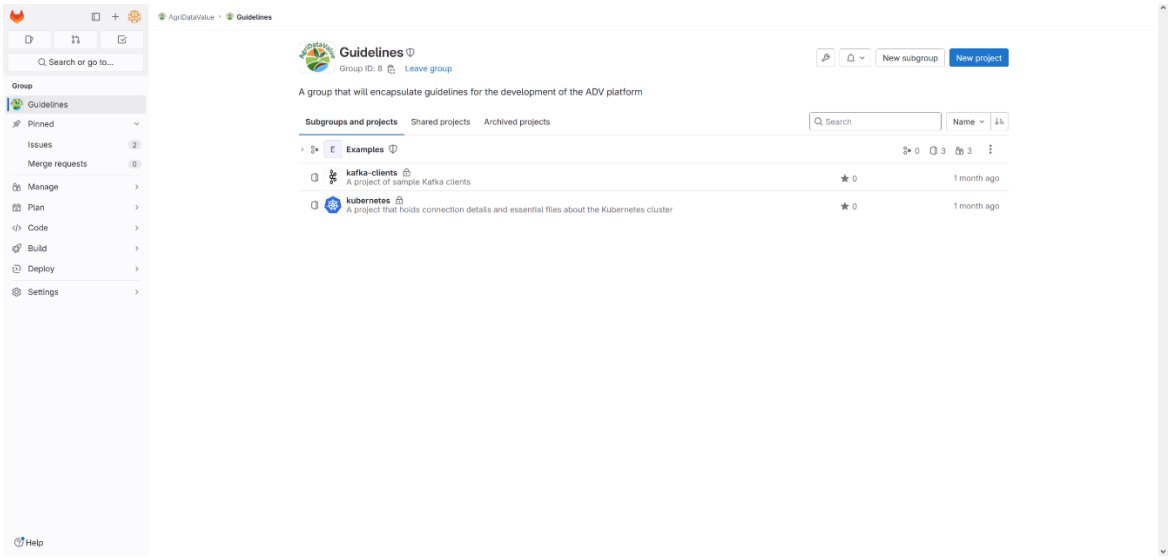


Figure 55: Documentation – Guidelines

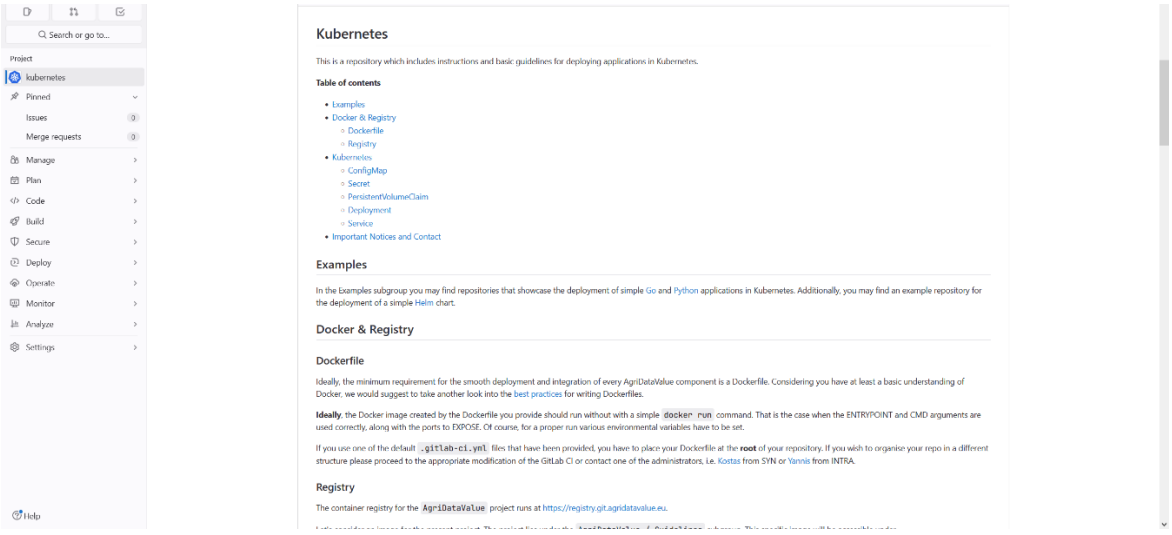


Figure 56: Documentation - Kubernetes example

8 Component verification and validation

The purpose of the Verification and Validation plan is to guarantee that AgriDataValue components are successfully integrated and operate as intended during the project's design phase. It outlines the procedure that AgriDataValue components must adhere to in order to undergo appropriate testing, and then describes the procedure that records and verifies their functional and non-functional performance both independently and as a component of a larger system (pilot).

8.1 Verification plan

The Verification Plan outlines the steps AgriDataValue implementations must take in order to:

- confirm that each application provides the functionality that each AgriDataValue integrated platform component was intended to provide during the design phase;
- confirm that the integration between each AgriDataValue integrated platform component has been successfully completed.

A series of Test Levels that can be applied on them help to realize this. In short, the goal of these tests is to confirm that every component:

- Can successfully integrate with the relevant AgriDataValue platform's components;
- Offers the necessary functionality for which it was designed;
- creates a system that satisfies the specified KPIs.

Test Levels are part of the verification plan, and they each cover a distinct aspect of the verification procedure. These are explained in the sections that follow.

The following test levels can be described in accordance with the Agile Test Extension of the International Software Testing Qualifications Board (ISTQB):

Component testing is often called unit, module, or program testing, and it looks for errors in software modules, programs, objects, classes, etc. that can be tested separately and verifies their functionality. It can be completely separately from the rest of the system, depending on the context of the development life cycle and the system itself. Differential component tests related to the development of the AgriDataValue integrated platform will be designed and implemented in each technical work package that supplies AgriDataValue components. Unit-test verification at the component level will be facilitated by these tests.

Integration testing evaluates the degree of interoperability between components and the degree of interoperability between systems. Systematic integration strategies can be built on top of system architecture (e.g., top-down and bottom-up), functional tasks, transaction processing sequences, or other aspects of the system or its components. Generally, integration should be incremental rather than a "big bang" to help isolate faults and find problems early.

System testing examines the general behaviour of a product. System environments should be as close to the production or final target environments as possible to minimize the chance that environment-specific defects will go undetected during testing. System testing may also be predicated on risks in addition to requirements specifications, business processes, use cases, and other high-level text descriptions or models of system behaviour, interactions with the operating system, and system resources. It is expected that the AgriDataValue Pilot leaders will oversee this testing phase, which ought to be organized ahead of the pilot demonstrations.

Acceptance testing aims to increase user confidence in the system, its parts, or some of its non-functional aspects. A system's users or customers are normally responsible for it; other stakeholders may also be engaged. Finding errors is not the main goal of acceptance testing. Although it's not always the last phase of testing, acceptance testing can assess how ready the system is for use and deployment. This level of testing is optional because it is not anticipated that the AgriDataValue platform will be commercialized within the project timeline.

Figure 57: AgriDataValue's Component Test Levels

Figure 57 illustrates, in a summative manner, the Test Levels that each component must be validated and verified against in the context of the AgriDataValue project.

8.2 Validation plan

The process that AgriDataValue pilot partners must adhere to in order to validate the various AgriDataValue components that are being used in their pilots is outlined in the Validation Plan.

Validation using storyboards. The release-based validation will be carried out by the pilot owners. The AgriDataValue platform will be validated during the pilot demonstration phases using a special release validation form that contains all functionalities requested by users for each pilot. The validation process will produce a list of features that are not implemented, partially implemented, or fully implemented. A list of the missing features and an analysis of the problem(s) will be provided to assist the relevant AgriDataValue partners in resolving the potential issue(s).

Documentation needs to be completed for each verified feature or component. The documentation for every feature and component will be gathered and examined by the pilot owners, who will follow a template for documentation. The missing documentation will be communicated to the relevant AgriDataValue partners. The documentation provided to the pilot leaders will serve as the foundation for the validation process.

Annex I outlines the necessary documentation that every owner of a component must possess. After every pilot-based validation, the verified documentation will be included in the platform release package.

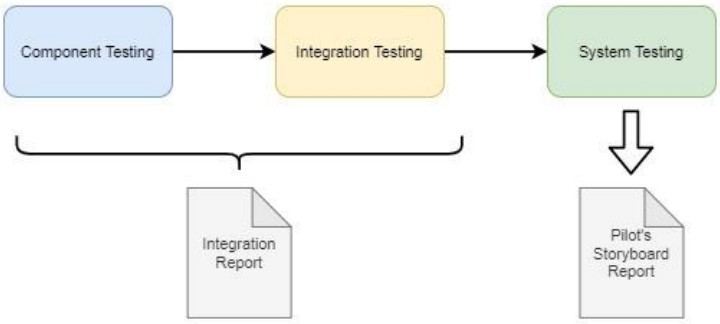


Figure 58: Verification and Validation process

Figure 58 above illustrates the Verification and Validation process that needs to be followed by each AgriDataValue partner. After conducting component and integration testing, each partner generates an integration report that compiles the testing results. Based on the component integration reports, pilot leaders

will produce a storyboard validation report attesting to the implementation of each pilot's requested functionalities.

Verify the KPIs, or key performance indicators. Each release will have a set of quantifiable and validated key performance indicators (KPIs), such as not reached, partially reached, and fully reached. To assist in achieving the goal, technical partners will receive a KPI validation report. The KPIs are included in the template that is described in Annex I.

8.3 Technical requirements – the update process

The technical requirements and technical specifications (initial version presented in D1.3) of AgriDataValue project are closely linked to the progress of the rest of the components detailed in the present document. This is also part of the verification report for each component as their functionality aims to cover the technical (and consequently the user) requirements. Due to the changing nature of the knowledge and the complexity of each component and for keeping an up-to-date status, an action plan has been established.

Table 1 - Mapping between Technical requirements and User requirements, as defined in D1.3 needs periodical verification, which is performed in the context of Task 1.4. Components which are targeted for the update are all components of the ADV platform, and together with the component owners' collaboration, updates are further to be identified or detailed.

Regarding the potential updates of the technical requirements, actions launched and aimed to be performed are:

- Iterate through the list of technical requirements and technical specifications from D1.3 with each partner for potential updates;
- Check if there are any user requirements which are not covered by any technical requirements and investigate whether they can be covered;
- Check if some of these requirements are no longer relevant, to conclude what is to be detailed further (brief description);
- Run an initial validation and verification through the pilots (verify what is covered, what is missing);

The expected result is an up-to-date set of technical requirements with the possibility of real-time changes based on current knowledge and AgriDataValue project'sdevelopment.

As an example, the procedure of updating technical requirements as defined for the "Satellite Earth Observation Data Capturing Toolbox" (EOD) component is presented. The initial technical requirements and user requirement mapping in D1.3 for this component was the following:

ID	Type	Title	Related component	Priority	Related User Requirement	Verification Method (Description/Demonstration of functionality, Achieved/Not Achieved, N/A)
Satellite Earth Observation Capturing requirements						
REQ.FN.08.01	Functional	EO imagery catalogue	EOD	S	REQ.US.01	
REQ.FN.08.02	Functional	EO imagery access	EOD	S	REQ.US.01	

ID	Type	Title	Related component	Priority	Related User Requirement	Verification Method (Description/Demonstration of functionality, Achieved/Not Achieved, N/A)
REQ.FN.08.03	Functional	Ingestion of and access to other datasets	EOD	S	REQ.US.01	
REQ.FN.08.04	Functional	EO data processing	EOD	S	REQ.US.01	
REQ.FN.08.05	Functional	EO Large Scale (raster) processing	EOD	S	REQ.US.01	
REQ.FN.08.06	Functional	EO Large Scale (object-based) processing	EOD	S	REQ.US.01	

Through the live technical requirements update procedure, five (5) more Technical Requirements were added for this component:

ID	Type	Title	Related component	Priority	Related User Requirement	Verification Method (Description/Demonstration of functionality, Achieved/Not Achieved, N/A)
Satellite Earth Observation Capturing requirements						
REQ.FN.08.07	Functional	Catalogue STAC compliance	EOD	S	REQ.US.01	
REQ.FN.08.08	Functional	OGC compliance	EOD	S	REQ.US.01	
REQ.FN.08.09	Functional	COG support	EOD	S	REQ.US.01	
REQ.FN.08.10	Functional	Sentinel data availability	EOD	S	REQ.US.01	
REQ.FN.08.11	Functional	Machine learning and AI integration	EOD	S	REQ.US.01	

8.4 Verification report collection status

The template provided in Annex I was completed by component owners whose components were included in the AgriDataValue integrated platform first version, as previously mentioned in the first two sections. Each component's functionality and integration tests were included in the reports. The EU has access to the comprehensive details of these documents, which are kept on AgriDataValue's online documentation platform (Owncloud, hosted by Synelixis). As of the submission of this deliverable, we have gathered seven (7) reports that cover WP2 components; the rest of the reports are in progress following the development timeline of their respective components. This is a summary of the status of the report collection; updated reports will be supplied as components are updated, and the collection is still ongoing.

9 Conclusions and Next Steps

The components, tools, and technologies of the first, baseline, version of the ADV integrated platform —which facilitates solution integration, platform interoperability, and pilot case support—are described in this document. These components, tools and technologies are made available in a manner that:

- provides an overview of the components forming the ADV integrated platform;
- offers a tangible implementation that the pilot applications can use to direct future development; and
- gives complete flexibility for application configuration and deployment to support the diverse needs of the stakeholders and the wide range of pilot needs.

The document utilizes and presents work mainly conducted within the context of WP2, as well as from technical work originating in the context of WP3 and WP4. It supports project Milestone 4 "Agri-environment Platform & Tools (ADS Baseline)" and offers the baseline release of the ADV integrated platform.

The ADV pilots will utilise and evaluate this release in the upcoming months, in order to develop and assess the ADV applications that will be leveraged during their execution. A revised version of this deliverable will be presented in D2.2 (M24) and D2.3 (M36), incorporating the feedback of the end users.

The development of the first, baseline, version of the integrated platform, the initial integration of the ADV components, and the verification and validation plan for the ADV components are the main accomplishments of D2.1.

As a next step, the ADV components and the integrated ADV platform, Version 1, will be presented in D2.2, which is scheduled for release in the beginning of 2025.

References

[1] Synelixis Solutions S.A., “SynField - The path toward smart agriculture,” [Online]. Available: https://www.synfield.gr/. [Accessed 12 2023].

[2] Synelixis Solutions S.A., “About SynAir - SynField,” [Online]. Available: https://www.synfield.gr/about-synair/. [Accessed 12 2023].

[3] R. N. &. S. P. R. Strange, “Plant disease: a threat to global food security,” in *Annu. Rev. Phytopathol*, 2005, pp. 43, 83-116.

[4] Q. L. Y. C. T. &. T. Yang, “Federated machine learning: Concept and applications.,” *ACM Transactions on Intelligent Systems and Technology (TIST),* vol. 10(2), pp. 1-19, 2019.

[5] Liu, L., Zhang, J., Song, S. H., & Letaief, K. B., “Client-edge-cloud hierarchical federated learning,” *ICC 2020-2020 IEEE International Conference on Communications (ICC),* pp. 1-6.

[6] Papernot, N., Song, S., Mironov, I., Raghunathan, A., Talwar, K., & Erlingsson, Ú., “Scalable private learning with pate.,” *arXiv preprint arXiv:1802.08908,* 2018.

[7] Abadi, M., Chu, A., Goodfellow, I., McMahan, H. B., Mironov, I., Talwar, K., & Zhang, L., “Deep learning with differential privacy.,” *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security,* pp. 308-318, 2016.

[8] Dwork, C., “Differential privacy.,” *International colloquium on automata, languages, and programming (pp. 1-12).,* vol. Springer Berlin Heidelberg., 2006.

[9] McMahan, B., Moore, E., Ramage, D., Hampson, S., & y Arcas, B. A., “Communication-efficient learning of deep networks from decentralized data.,” *Artificial intelligence and statistics,* pp. 1273-1282, 2017.

[10] Martín Abadi, et. al,, “Tensorflow: Large-scale machine learnin on heterogeneous systems,” 2015.

[11] IETF OAuth Working Group, “OAuth 2.0,” [Online]. Available: https://oauth.net/2/.

[12] OpenID, “Welcome to OpenID Connect,” 2020. [Online]. Available: https://openid.net/connect/.

[13] OASIS, “SAML Version 2.0 Errata 05,” 2012. [Online]. Available: http://docs.oasis-open.org/security/saml/v2.0/sstc-saml-approved-errata-2.0.html.

[14] Zhao, Y., Chen, J., Zhang, J., Wu, D., Blumenstein, M., & Yu, S., “Detecting and mitigating poisoning attacks in federated learning using generative adversarial networks.,” *Concurrency and Computation: Practice and Experience,* vol. 34(7), 2022.

[15] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville and Y. Bengio, “Generative Adversarial Nets,” *Advances in neural information processing systems,* 2014.

Annex I

Kafka message schema

Generic-message schema

```
{
  "$schema": "http://json-schema.org/draft-07/schema",
  "$id": "http://eu-agridatavalue.com/v1/generic-message.schema.json",
        "title": "ADV Generic Message",
        "description": "Common message structure",
  "type": "object",
        "properties": {
                "header": { "$ref": "generic-header.schema.json" },
                "body":   { "$ref": "specific-body.schema.json" }
        },
        "required": [
                "header"
        ],
        "additionalProperties": false
}
```

Generic-header schema

```
{
  "$schema": "http://json-schema.org/draft-07/schema",
  "$id": "http://eu-agridatavalue.com/v1/generic-header.schema.json",
        "title": "ADV Generic Message-Header",
        "description": "Common header used by all messages",
  "type": "object",
  "version": "0.1",
        "properties": {
                "topicName": {
                        "description": "Message hub topic-name",
                        "type": "string"
                },
                "sender": {
                        "description": "Message sender identifier, e.g., INTRA",
                        "type": "string"
                },
                "sentUtc": {
                        "description": "Message origination time/date in UTC time (GMT)",
                        "type": "string",
                        "format": "date-time"
                },
                "source": {
                        "description": "Specific sender (component, subcomponent, etc.) identifier (e.g. STORE)",
                        "type": "string"
                },
                "messageId": {
                        "description": "A uid that uniquely identifies this message. It should be created by the
sender so that the receiver can use this in case a response is needed",
                        "type": "string"
                },
                "processId": {
                        "description": "A uid that correlates this message to a higher level process. It should be
created by the component where the process was initiated. This uid will add a higher-level perspective to the message",
                        "type": "string"
                }
        },
        "required": [
```

```
                    "topicName",
                    "sender",
                    "sentUtc",
                    "source"
            ],
            "additionalProperties": true
}
```

Generic-body schema

```
{
   "$schema": "http://json-schema.org/draft-07/schema",
   "$id": "http://eu-agridatavalue.com/v1/specific-body.schema.json",
            "title": "ADV Specific Message-Body",
            "description": "Specific body per topic in message-hub",
   "type": "object",
            "properties": {
                    "prop1": {
                            "description": "Sample property 1",
                            "type": "string"
                    },
                    "prop2": {
                            "description": "Sample property 2",
                            "type": "number"
                    }
            },
            "required": [
                    "prop1"
            ],
            "additionalProperties": true
}
```

Component verification report template

Component Validation Report

Table 14, below tabulates the general information of an AgriDataValue component that is deployed and validated. Some information is already available in the component information spreadsheet, so you can copy it from there.

Component general description

Table 14: Component's general description

Title	This field holds the name of the AgriDataValue component	WP	This field holds the WP that the component belongs to
Component Identifier	This field holds a short identifier for the component (e.g., DKM)		
Description	This field holds the component's operation description		
Repository type	This field holds the repository type of the source code of the component. (e.g., Private, AgriDataValue GitLab)	Justification	This field holds the justification of the source code repository type selection

Repository URL	If the Repository type is in AgriDataValue 's GitLab, then the absolute URL of the component's location must be filled in here.
Features' list	The list of the functionalities provided by the component in a bullet list
Integration component list	This field holds the list of components that this component interoperates and will integrate with (use the component information spreadsheet for this)
Deployment location	This field holds deployment location (e.g., AgriDataValue cloud infrastructure, pilot premises, proprietary location, etc)
Docker image location and size	If the component is containerized, then please fill in the location of the image registry that resides and the size of the docker image
Requirements	This field holds computational requirements for this component. Among others, you can describe here the CPU, RAM, STORAGE requirements of the component.
Contact email	This field holds the email of the developer (or contact person) of the component.

Integration and Functionality Tests

Integration tests verify that your code works with external dependencies correctly. Functionality tests verify that your component sufficiently covers the features needed, as expressed in previously defined requirements. Tests shall be described and include/adhere the following categories:

1. **Integration Tests:** Validating the interconnection between the components to be integrated. Provide in a tabulated format the summary of the integration activities performed. Indicate whether an integration activity is successful or not successful. You should also indicate the Pilot/infrastructure that where these tests were executed.

Test ID	**<COMPONENT_IDENTIFIER_ITCXX> (e.g., < DKM _ITC01>)**
Test description	
Component(s) under test	(Related to the Integration component list)
Component(s) under test environment	(where each component is deployed for the integration test, e.g., lab, AgriDataValue testing env, pilot production env, etc)
Dependencies	(Any dependencies on other components, environments, etc)
Steps	1. Step 1 2. Step 2 3. …
Pass criteria	(What qualifies this test as passed)
Result	(Pass / No Pass)
Comments	(any details/comments, e.g., in the case that the test did not pass)

2. **Functionality Tests:** Validating the functionality provided by the component. Test all the test cases for the component. Provide in a tabulated format the summary of the functionalities provided by the

component. Indicate, whether a specific functionality (feature) is fully implemented, partially implemented, or not implemented.

Test ID	**<COMPONENT_IDENTIFIER_FTCXX> (e.g., <DKM_FTC01>)**
Test description	
Related technical requirement(s)	(Have a look at Appendix I)
Feature(s) under test	(Related to the Component feature list)
Test environment	(e.g., lab, AgriDataValue testing env, pilot production env, etc)
Dependencies	(Any dependencies on other components, environments, etc)
Steps	1. Step 1 2. Step 2 3. …
Pass criteria	(What qualifies this test as passed)
Result	(Pass / No Pass)
Comments	(any details/comments, e.g., in the case that the test did not pass)

Validation summary

In this section, each partner shall include a summary of the performed functionality and integration tests for this component (how many tests per type have been performed, how many were passed, were there any failed tests, what was the coverage of the functionality, which Technical Requirements where covered, etc). Please also include any useful comments about the testing and the integration process. Integration is considered successful when:

- The previous descriptions have been submitted.
- Sufficient functionality and integration tests have been carried out providing adequate coverage of the functionality provided by each component and interconnection with other AgriDataValue platform components.

Component KPIs

In this section, each partner will need to include component KPIs (e.g., System performance), *if applicable*. This section **does not** concern business KPIs or KPIs included in the GA, but technical-oriented ones that are specific for their component.

KPI	**Name**	**Description**	**Metric**	**Method of measurement**	**Target**	**Result**
System performance						
<KPI_COMPONENT_IDENTIFIER_XX> (e.g., KPI_DKM_01)	(the name of the KPI, e.g., Model update time)	(description of the KPI, e.g., time to update a model after fetching new data)	(e.g., Time in seconds)	(method to measure if the KPI was reached, e.g., measure the total time required to update a model)	(KPI target, e.g., <1min)	**TO BE FILLED AND UPDATED FOR DELIVERABLES D2.2/3**

EOD API description

Title	*The Processing API (or shortly "Process API") is the most commonly used API in Sentinel Hub as it provides images based on satellite data. Users can request raw satellite data, simple band combinations such as false colour composites, calculations of simple remote sensing indices like NDVI, or more advanced processing such as calculation of Leaf area index (LAI).*
URL: api/v1/process	
Full documentation also available here: https://docs.sentinel-hub.com/api/latest/api/process/	
Method *This field holds the type of the Method used*	
POST	
URL Params *This field holds the parameters (if any). Separated based on the fields below into required and optional.*	
/	
Data Params *This field holds the body payload of a post request.*	
Required:	
input	• *bounds: Defines the request bounds by specifying the bounding box and/or geometry for the request. If both are specified it will generate an image for the bounding box and render data contained within the geometry.* • *data: The collections you wish to request, along with certain processing and filtering parameters.*
output	*width: The request image width. Must be an integer between 1 and 2500.* • *Height: The request image height. Must be an integer between 1 and 2500.* • *Resx: Spatial resolution of the request image in a horizontal direction.* • *Resy: Spatial resolution of the request image in a vertical direction.* • *Responses: Response object(s).*
Evalscript	*The user provided evalscript*[59] .
Optional:	
image_id=[alphanumeric]	*parameter description*
Success response <What should the status code be on success and is there any returned data? This is useful when people need to know what their callbacks should expect>	
200 Content: image/jpeg or image/png or image/tiff	*Returns the imagery as requested by the user.*
Error response *This field holds the list of all possible error responses. Doing that, helps prevent assumptions of why the endpoint fails and saves a lot of time during the integration process.*	
400 Content: application/json	*Bad request*
500 Content: application/json	*Server error*
Sample call *This field holds a possible sample call to the described endpoint in a curl-like format. Please, choose the format wisely so that is clear and easy to read by the interested parties.*	
`import requests`	

[59] https://docs.sentinel-hub.com/api/latest/evalscript/v3/

```
response = requests.post('https://services.sentinel-hub.com/api/v1/process',
  headers={"Authorization" : "Bearer <your_access_token>"},
  json={
    "input": {
        "bounds": {
            "bbox": [
                13.822174072265625,
                45.85080395917834,
                14.55963134765625,
                46.29191774991382
            ]
        },
        "data": [{
            "type": "sentinel-2-l2a"
        }]
    },
    "evalscript": """
    //VERSION=3

    function setup() {
      return {
        input: ["B02", "B03", "B04"],
        output: {
          bands: 3
        }
      };
    }

    function evaluatePixel(
      sample,
      scenes,
      inputMetadata,
      customData,
      outputMetadata
    ) {
      return [2.5 * sample.B04, 2.5 * sample.B03, 2.5 * sample.B02];
    }
    """
})
```

Notes *This field holds any additional helpful info related to this endpoint.*

Title	*Batch Processing API (or shortly "Batch API") enables you to request data for large areas and/or longer time periods for any Sentinel Hub supported collection, including BYOC (bring your own data).*
URL: api/v1/batch/process	
Full documentation also available here: https://docs.sentinel-hub.com/api/latest/api/batch/	
Method *This field holds the type of the Method used*	
POST	
URL Params *This field holds the parameters (if any). Separated based on the fields below into required and optional.*	
/	

Data Params *This field holds the body payload of a post request.*	
Required:	
processRequest.input	• *bounds: Defines the request bounds by specifying the bounding box and/or geometry for the request. If both are specified it will generate an image for the bounding box and render data contained within the geometry.* • *data: The collections you wish to request, along with certain processing and filtering parameters.*
processRequest.output	*width: The request image width. Must be an integer between 1 and 2500.* • *Height: The request image height. Must be an integer between 1 and 2500.* • *Resx: Spatial resolution of the request image in a horizontal direction.* • *Resy: Spatial resolution of the request image in a vertical direction.* • *Responses: Response object(s).*
processRequest.evalscript	*The user provided evalscript*[60] .
Optional:	
output	• *defaultTilePath: Path or path template specifying where batch processing results shall be stored* • *overwrite: If true, the request will never fail if files already exist. Instead, any existing files will be overwritten, except if skipExisting is true and all outputs for a tile exist.* • *skipExisting: If true, any tiles for which all outputs already exist will be skipped.* • *cogOutput: If true, the results will be written as COG (cloud optimized GeoTIFFs).* • *collectionId: If provided, the results will be written as COG (cloud optimized* GeoTIFFs) and added to the existing collection with the specified identifier.
bucketName	Simplified alternative for specifying where the results shall be written, where only the bucket name is specified.
zarrOutput	Specifies Zarr creation parameters. If this parameter is specified, all outputs in processRequest must be of the type zarr/array and neither bucketName nor output can be specified.
Success response <What should the status code be on success and is there any returned data? This is useful when people need to know what their callbacks should expect>	
200 Content: application/json	*Returns the metadata information of the request.*
Error response *This field holds the list of all possible error responses. Doing that, helps prevent assumptions of why the endpoint fails and saves a lot of time during the integration process.*	
400 Content: application/json	*Unauthorized*
401 Content: application/json	*Unauthorized*
403 Content: application/json	*Insufficient permissions*

[60] https://docs.sentinel-hub.com/api/latest/evalscript/v3/

Sample call *This field holds a possible sample call to the described endpoint in a curl-like format. Please, choose the format wisely so that is clear and easy to read by the interested parties.*
See documentation for example : https://docs.sentinel-hub.com/api/latest/reference/#tag/batch_process
Notes *This field holds any additional helpful info related to this endpoint.*

Title	*The Statistical API (or shortly "Stats API") enables you to get statistics calculated based on satellite imagery without having to download images. In your Statistical API request, you can specify your area of interest, time period, evalscript and which statistical measures should be calculated. The requested statistics are returned in the API response*
URL: api/v1/statistics	
Full documentation also available here: https://docs.sentinel-hub.com/api/latest/api/statistical/	
Method *This field holds the type of the Method used*	
POST	
URL Params *This field holds the parameters (if any). Separated based on the fields below into required and optional.*	
/	
Data Params *This field holds the body payload of a post request.*	
Required:	
input	• *bounds: Defines the request bounds by specifying the bounding box and/or geometry for the request. If both are specified it will generate an image for the bounding box and render data contained within the geometry.* • *data: The collections you wish to request, along with certain processing and filtering parameters.*
aggregation	• *timeRange: Time range for which we want statistics to be computed* • *aggregationInterval: S*pecifies how given time range is split into time intervals • *width:* Width of the sample matrix. • *Height:* Height of the sample matrix. • *Resx:* Spatial resolution used to calculate the height of the sample matrix from the input.bounds. • *Resy:* Spatial resolution used to calculate the height of the sample matrix from the input.bounds
Evalscript	*The user provided evalscript*[61].
Optional:	
calculations	Define which statistics and histogram to calculate. It can be specified differently for each evalscript output. If omitted only the basic statistic (min, max, mean, stDev) will be calculated.
Success response <What should the status code be on success and is there any returned data? This is useful when people need to know what their callbacks should expect>	
200 Content: application/json	Statistics for intervals, where data is available
Error response *This field holds the list of all possible error responses. Doing that, helps prevent assumptions of why the endpoint fails and saves a lot of time during the integration process.*	
400	*Bad request*

[61] https://docs.sentinel-hub.com/api/latest/evalscript/v3/

Content: application/json	
403 Content: application/json	*Insufficient permissions*
Sample call *This field holds a possible sample call to the described endpoint in a curl-like format. Please, choose the format wisely so that is clear and easy to read by the interested parties.*	
Sample call is available on the documentation page: https://docs.sentinel-hub.com/api/latest/reference/#tag/statistical	
Notes *This field holds any additional helpful info related to this endpoint.*	

Fleviden Framework

Introduction

FL is a distributed approach of ML that trains a ML model across multiple decentralized edge devices or clients holding data without exchanging it. Fleviden, which stands for Federated Learning Eviden, is a fully extensible distributed learning framework originally developed internally by the Research and Development department of Atos. It has been included as part of the background technologies described in the project Consortium Agreement (CA).

Within the AgriDataValue project, we have the specific objective of extending and improving this asset to handle privacy-preserving HFL. This improvement is related to the main objectives identified for task 2.2 edge-driven analytics / FDML:

1. To build an efficient and scalable FL framework to train DL models for the use cases in a federated way.
2. Enhance the privacy of the FDML with privacy-preserving techniques such as PATE.

The following subsection (“Background for Fleviden Framework) is mostly based on the background developed by ATOS for Fleviden and is provided here for the sake of comprehensiveness of the document while section “Hierachical Federated Learning Agents in Fleviden) is devoted to the aspects that have been specifically implemented in the AgriDataValue project.

Background for Fleviden Framework

Fleviden is the Python framework for Federated and Distributed Learning developed by Atos. Following the principle of divide and conquer, Fleviden proposes a modular architecture for the development of distributed ML based on *pipes and filters*, embodied by its basic unit called *pod*. A *pod* defines several input and output interfaces and receives several inputs, performs a specific atomic functionality based on those inputs, and triggers the appropriate output based on it. The concatenation of different pods allows the generation of applications flexibly and efficiently.

This modular architecture proposed by Fleviden offers some advantages that are essential for the AgriDataValue project:

1. **Flexibility**: compared to other frameworks, facilitates the implementation of more complex architectures, such as the HFL that is intended to be implemented in task 2.2.

2. **Interoperability**: Fleviden is fully decentralized and can be deployed on-premises and cross-cloud infrastructure. It can even be deployed in a hybrid manner and utilize recent paradigms like server-less computation.

3. **Privacy-preserving**: Fleviden currently supports privacy-preserving approaches such as secure sum and in-plan homomorphic encryption and differential privacy. This is a key objective for task 2.2. Additional solutions like Multiparty Computation and PATE can be extended.

In addition to these advantages, we remark the fact that this technology is owned by a consortium partner enables the framework to be tailored to the project's needs, rather than the other way around. Furthermore, it ensures the maintenance of the framework even after the project's completion.

Hierarchical Federated Learning Agents in Fleviden

In this section we delve into the implementation of the diverse FL Agents required in the AgriDataValue project that are located inside FDML and DKM components.

DKM

The DKM performs the functionality associated with the root server of the HFL. In the Figure 59, the implementation of this component is described in terms of Fleviden pods.

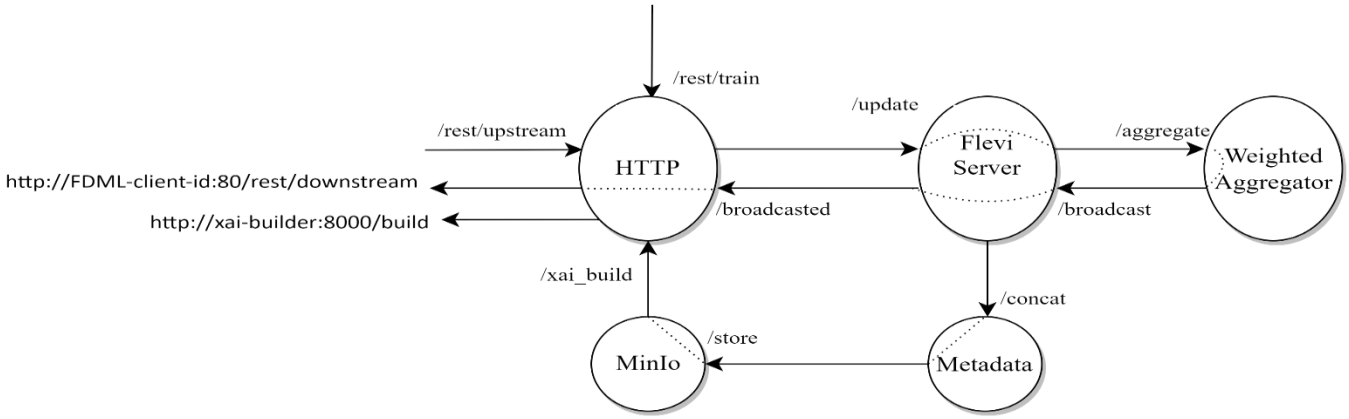


Figure 59. Fleviden architecture for the development of the DKM component

Five pods are involved in the DKM:

1. **HTTP Pod:** This communication pod exposes an API REST with the following endpoints or input interfaces:
    a. */rest/train:* triggering this interface starts the HFL process. It receives as input the metadata defined by the end user in a JSON format. This metadata includes the domain and pilot information recovered in Table 1.
    b. */rest/upstream:* input interface where the local models from the FDML component are received in a JSON format.

    Additionally, this pod triggers two external interfaces:
    c. http://xai-builder:8000/build input interface of the XAI sub-component dedicated to training the XAI explainer associated with the global model. This interface is triggered at the end of the training process with the metadata defined during the process in a JSON format.
    d. http://FDML-client-id:80/rest/downstream input interface of the FDML agents linked to the DKM. This interface is triggered at the end of each round with the weights of the global model generated in a JSON format.
2. **Fleviden server:** This pod orchestrates the cross-silo FL:
    a. It is initialized with a list of associated clients. This list of clients can be defined directly in the Docker compose that deploys the component.

   b. Each time it receives a local model, it performs the authentication process described in section 6.1. All the weights that pass through the authentication process are sent to the aggregator pod to conform the global model.
   c. It checks the actual round of the FL process. When the final round is reached it triggers the Metadata pod to finish the learning process.
3. **Weighted aggregator:** This pod receives the local models' weights in a JSON format through its interface */aggregate.* When a specific number of models from different clients are received, the aggregator performs the FedAvg technique.
4. **Metadata:** This pod generates the metadata described in Table 1 and concatenates it to the message received through its input interface */concat.* In this way, the final message/output is a JSON with the global model weights and the associated metadata.
5. **MinIO Pod:** This pod receives as input the weights of the model and the metadata associated with the global model through its input interface */store.* This pod generates the global model in a *pickle* format and stores it in the SECURESTORE MinIO bucket *models.* Additionally, the pod saves the metadata associated with the model in JSON format.

FDML Intermediate Server

The intermediate server has been developed using the architecture shown in Figure 60.

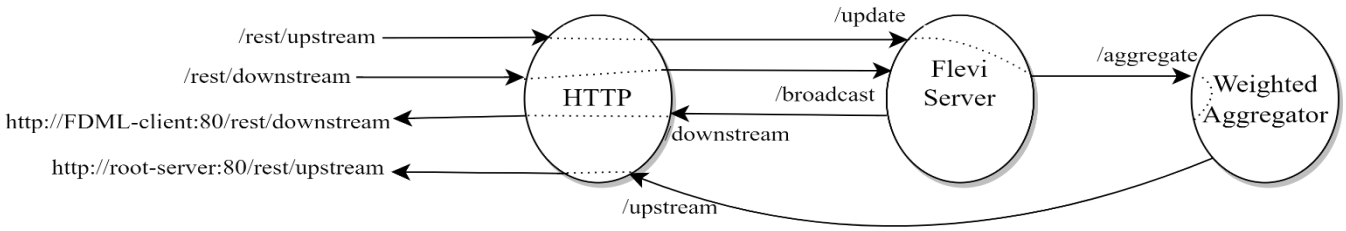


Figure 60. Fleviden architecture for the development of the FDML intermediate server sub-component

As seen in the previous figure, the pods involved in the architecture of the FDML intermediate servers have already been described for the development of the DKM. However, the connections defined between the input and output interfaces make the final functionality slightly different. The functionality of the intermediate server can be divided into two flows depending on the input interface of the pod HTTP triggered:

1. */rest/upstream:* Through this interface, the FDML Intermediate servers receive the local models in a JSON format. The weights are forwarded until the weighted aggregator generates the regional models. The aggregated model is forwarded to the root server in the DKM through the pod HTTP by the interface: http//root-server:80/rest/upstream.
2. */rest/downstream:* Through this interface, the Intermediate server receives the weights from the global model generated by the DKM. Then these weights are broadcasted to the FDML clients by the interface http://FDML-client:80/rest/downstream.

FDML client with Fleviden

Figure 61 represents the Fleviden architecture implemented for the development of a FDML Client.

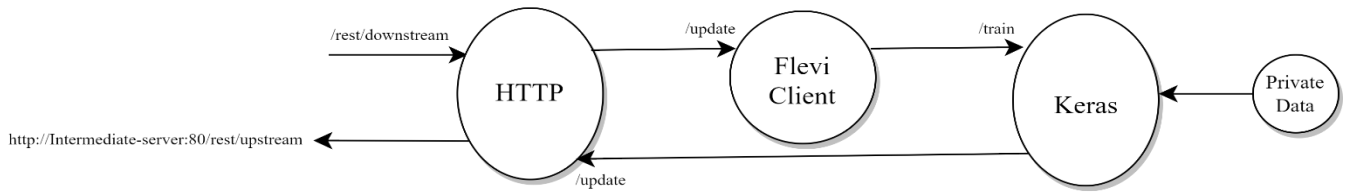


Figure 61. Fleviden architecture for the development of an FDML Client

The pods involved in this architecture are:

1. **HTTP pod:** This communication pod exposes an API REST with */rest/downstream* input interface or endpoint. This interface is triggered with the global model received from the intermediate servers of the FDML component in a JSON format. Additionally, the local updates of the model generated by the Keras pod are forwarded in JSON format to the endpoint *http://Intermediate-server:80/rest/upstream*
2. **Flevi Client**: This pod orchestrates the FL on the client side.
3. **Keras pod:** This pod receives as input a serialized model in a JSON format and trains it using the Keras framework. To do this, this pod extracts the previously pre-processed private data. It is important to remark that in this version of the platform, the private data are located in the clients in *CSV format*. However, in future implementations we will include the possibility to access the data to an API and the pre-processing dedicated for each data type will be encapsulated in a Fleviden pod.