D4.4

Final report and release on cross-border application data management and use-case specific AI solutions

30/03/2021
The DECENTER Consortium has addressed all comments received, making changes as necessary. Changes to this document are detailed in the change log table below.

<table>
<thead>
<tr>
<th>Date</th>
<th>Edited by</th>
<th>Status</th>
<th>Changes made</th>
</tr>
</thead>
<tbody>
<tr>
<td>11/11/2020</td>
<td>Petar Kochovski</td>
<td>Plan</td>
<td>Table of Contents</td>
</tr>
<tr>
<td>25/11/2020</td>
<td>Petar Kochovski, Sungwoo Kum</td>
<td>Plan</td>
<td>Revised Table of Contents</td>
</tr>
<tr>
<td>8/12/2020</td>
<td>Petar Kochovski</td>
<td>Working</td>
<td>Added Section 2 content</td>
</tr>
<tr>
<td>11/12/2020</td>
<td>Seungwoo Kum</td>
<td>Working</td>
<td>Added contributions in Section 2</td>
</tr>
<tr>
<td>14/12/2020</td>
<td>Jaewon Moon</td>
<td>Working</td>
<td>Added contributions in Section 7</td>
</tr>
<tr>
<td>16/12/2020</td>
<td>Adrian Arroyo Perez</td>
<td>Working</td>
<td>Added Section 5 content</td>
</tr>
<tr>
<td>17/12/2020</td>
<td>Seungwoo Kum</td>
<td>Working</td>
<td>Added contributions in Section 5 and 6</td>
</tr>
<tr>
<td>21/12/2020</td>
<td>Petar Kochovski</td>
<td>Working</td>
<td>Added contributions in Sections 3 and 4</td>
</tr>
<tr>
<td>25/12/2020</td>
<td>Janez Brežnik, Petar Kochovski</td>
<td>Draft</td>
<td>Content consolidation</td>
</tr>
<tr>
<td>30/12/2020</td>
<td>Vlado Stankovski</td>
<td>Final</td>
<td>Final draft</td>
</tr>
<tr>
<td>1/02/2021</td>
<td>Petar Kochovski</td>
<td>Working</td>
<td>Update Section 4</td>
</tr>
<tr>
<td>15/02/2021</td>
<td>Petar Kochovski</td>
<td>Working</td>
<td>Update Section 5</td>
</tr>
<tr>
<td>20/02/2021</td>
<td>Seungwoo Kum</td>
<td>Working</td>
<td>Extend Section 4</td>
</tr>
<tr>
<td>28/02/2021</td>
<td>Seungwoo Kum</td>
<td>Working</td>
<td>Update Section 8</td>
</tr>
<tr>
<td>22/03/2021</td>
<td>Seok Hee Lee</td>
<td>Working</td>
<td>Internal review</td>
</tr>
<tr>
<td>25/03/2021</td>
<td>Raffaele Giaffreda</td>
<td>Working</td>
<td>Internal review</td>
</tr>
<tr>
<td>26/03/2021</td>
<td>Petar Kochovski, Seungwoo Kum, Jaewon Moon</td>
<td>Working</td>
<td>Addressing internal review comments</td>
</tr>
<tr>
<td>30/03/2021</td>
<td>Vlado Stankovski</td>
<td>Final</td>
<td>Final draft</td>
</tr>
</tbody>
</table>

Notice that other documents may supersede this document. A list of latest public DECENTER deliverables can be found at the DECENTER Web page at https://www.decenter-project.eu/.

**Copyright**

This report is © DECENTER Consortium 2020. Its duplication is restricted to the personal use within the Consortium, funding agency and project reviewers.
Acknowledgements

This project has received funding from the European Union’s Horizon 2020 Research and Innovation Programme under grant agreement no. 815141 (DECENTER: Decentralised technologies for orchestrated Cloud-to-Edge intelligence).

This work was supported by Institute for Information & communications Technology Promotion (IITP) grant funded by the Korea government (MSIT/IITP) (No. 1711075689, Decentralised cloud technologies for edge/IoT integration in support of AI applications).

The partners in the project are FONDAZIONE BRUNO KESSLER (FBK), ATOS (ATOS), COMMISSARIAT À L’ENERGIE ATOMIQUE ET AUX ENERGIES ALTERNATIVES (CEA), COMUNE DI TRENTO (TN), ROBOTNIK (ROB), UNIVERZA V LJUBLJANI (UL), KOREA ELECTRONICS TECHNOLOGY INSTITUTE (KETI), GLUESYS (GLSYS), DALIWORKS (DW), LG U+ (LGUP), SEOUL NATIONAL UNIVERSITY (SNU).

The content of this document is the result of extensive discussions within the DECENTER © Consortium as a whole.

More information

Public DECENTER reports and other information pertaining to the project are available through DECENTER public Web site under http://www.decenter-project.eu.
Contents

Revision history ..................................................................................................................... 2
Copyright .............................................................................................................................. 2
Acknowledgements ............................................................................................................... 3
More information ................................................................................................................... 3
List of figures ........................................................................................................................ 6
List of tables .......................................................................................................................... 7
Executive Summary .............................................................................................................. 8
1  Introduction .................................................................................................................... 9
2  Background ..................................................................................................................... 10
   2.1 Blockchain, Smart Contracts and Smart Oracles ................................................... 10
      2.1.1 Blockchain .................................................................................................... 10
      2.1.2 Smart Contracts ........................................................................................... 12
      2.1.3 Smart Oracles .............................................................................................. 13
   2.2 AI Model Management and Related Techniques ................................................... 13
      2.2.1 Libraries and repositories for easy use of AI models .................................... 13
      2.2.2 AI model as Data .......................................................................................... 16
3  Trust Management for AI Based Fog Computing Applications ................................ 18
   3.1 Definition of trust in smart environments ................................................................ 18
   3.2 Relevant trust attributes ......................................................................................... 19
4  Multi-Party Service Level Agreements for Multi-Party Data Governance .................. 21
   4.1 Architecture for multi-party SLA management ....................................................... 21
   4.2 Experimental evaluation ........................................................................................ 23
   4.3 Summary ............................................................................................................... 24
5  AI Model (Data) Management Operations and Architecture ....................................... 25
   5.1 GDPR and PIPA regulations .................................................................................. 25
      5.1.1 GDPR ........................................................................................................... 25
      5.1.2 PIPA ............................................................................................................. 27
   5.2 Data Management Usage Scenario in compliance with GDPR/PIPA ..................... 28
6  Implementation of the cross-border AI model (data) management scenario ................. 30
   6.1 Architecture and implementation of AI Model (Data) Governance and Management 30
7  AI Model Repository final design, implementation, and integration ............................ 32
   7.1 AI model repository .............................................................................................. 32
      7.1.1 AI model repository structure ........................................................................ 32
      7.1.2 Semantics of the main elements .................................................................... 34
   7.2 Implementation environment .................................................................................... 38
7.2.1 Implementation environment ................................................................. 38
7.2.2 REST API Interfaces for AI model access ................................................ 38
7.2.3 Implementation results ........................................................................... 40
7.1 AI model repository on model exchange management for Federated Learning ... 41
  7.1.1 Storage in existing FL system ................................................................. 41
  7.1.2 The system architecture of AI model repository for FL ......................... 42
  7.1.3 The way to use the proposed repository for FL ..................................... 43
8 Kubeflow and DECENTER Model Serving approaches .................................... 45
  8.1 Kubeflow ........................................................................................................ 45
    8.1.1 Training workflow ................................................................................ 46
    8.1.2 Kubeflow as a Model Serving platform ................................................ 50
  8.2 DECENTER model serving ....................................................................... 51
9 DECENTER AI solutions and implementation guidelines ............................. 54
  9.1 Guideline to use DECENTER facilities .................................................... 55
10 Conclusion ........................................................................................................ 63
References ......................................................................................................... 64
Abbreviations ..................................................................................................... 66
List of figures

Figure 1. Training and deployment environment of Tensorflow [15] ........................................ 15
Figure 2. Torch Serve Process [16] ....................................................................................... 15
Figure 3. Architecture for trusted multi-party SLA management ........................................... 22
Figure 4. Gas consumption of all SC functions in SCs “A” and “B” ........................................ 23
Figure 5 Data Management usage scenario ........................................................................... 29
Figure 6 Architecture for Data Management and Governance .............................................. 30
Figure 7 Interaction with other Platform Components ............................................................ 31
Figure 8. Two different ways to use the AI model on the edge side .................................... 33
Figure 9. The folder structure of the model repository. ......................................................... 33
Figure 10. The basic structure of the AI model data type ......................................................... 35
Figure 11. Overall structure of model_information type .......................................................... 37
Figure 12. Registered AI Model Distribution ........................................................................ 41
Figure 13. The summary information of registered AI models .............................................. 41
Figure 14: Volume Creation Step in Pipeline from Kubeflow .............................................. 47
Figure 15: Training step of an AI Package model in Kubeflow Platform ................................ 47
Figure 16: Run of pipeline with logs of training an AI model .............................................. 48
Figure 17: Application components ....................................................................................... 48
Figure 18: The graph view for the Parallel pipeline ............................................................... 49
Figure 19: The graph view for the MV-sequential pipeline .................................................... 49
Figure 20: Technology stack of KFServing, the supported multi-framework model serving system of Kubeflow (extracted from KFServing GitHub Repository) .......................... 50
Figure 21: Software stack for AI application and virtualization ......................................... 51
List of tables

Table 1 Performance analysis for registration, deployment, and redeployment operations . 24
Table 2. AI models used to deploy in DECENTER for the first year of the project .............. 38
Table 3: Execution time (mm:ss) ......................................................................................... 49
Table 4: Kubeflow Serving, Sheldon Core and DECENTER Serving Supported features.... 52
Table 5: Use-case specific AI solutions’ designs................................................................... 54
Executive Summary

The essential idea in this deliverable is to describe the updated mechanisms for regulated, policy based cross-border data management and the final Use-case specific AI solution designs.

Within DECENTER, the cross-border data management scenario assumes that the Big Data pipeline may start at one service provider (e.g., camera owned by the municipality of Trento, Italy) and can proceed through secure (e.g., encrypted) Internet channels towards the other processing application stages, which may be implemented in other administrative domains, for example, private or public Cloud providers in Slovenia, and extended further to Cloud providers in Seoul, Korea. Hence, the term “border” in the context of our work refers to any administrative, organisational policy or government regulation in which data should pass. As a result, the process must abide to various specific policies and regulations, requirements for certification, permissions, including personal permissions and preferences.

Requirements extracted by specific parties taking part in a data management scenario may be too hard in which case it may prove impossible to establish the required quality of cross-border data management and transport. For example, regulations differ between countries by requiring collaboration with different cloud providers, implementation of different hardware resources, implementation of different security protocols, empowering people with different rights and similar. Thus, the cross-border data management mechanisms must be designed in a way to allow strict assessment and application of the user’s preferences in the specific context where the use of AI will be required.

This work presents the outcomes of tasks T4.4 and T4.5. Hence, this work has three goals: (1) present the outcome of the analysis, design and deployment of specific cross-border data management mechanisms that enable the participating entities control all aspects of the data transport and management when it comes to their administrative domains; (2) present the final design and implementation details of the AI model repository; (3) present how to configure and use the mechanism for injection of selected AI model from the DECENTER AI repository to the running AI container.

In this deliverable, we present the final design and implementation of the cross-border data management use case scenario that focuses on a personalised AI model, which is stored in a repository of models in Korea. The repository, designed by KETI, can be used to store sensitive (private) AI models. KETI also implemented mechanisms for injection of a selected AI model from the repository to the running AI (method) container. Based on this functionality, UL designed the cross-border management tasks, which involve cross-border movement of the sensitive AI model, that is, across regulatory and organisation borders. In addition to this, UL experimented with the design of several Smart Contracts that include the use of Smart Oracles to implement (1) full and shared governance of the AI model transport, (2) various monetisation scenarios that could be used in scenarios where the AI models are deployed for temporary use, (3) translation of GDPR regulations into the Smart Contracts logic. In addition, we designed, developed, and evaluated an exploratory scenario to investigate the ability to manage trust as a high-level property required to realise AI-based fog computing applications, and the ability to develop multi-party smart contracts for multi-party data governance in the cross-border management scenario.

Regarding the AI solutions, they were specifically designed and adopted by the DECENTER use cases. Each use case-specific solution was designed in such a way that the learning model hosted in the Cloud will be periodically updated and enhanced based on the feedback received from the AI model repository.
1 Introduction

This deliverable describes the DECENTER Fog Computing and Brokerage Platform's mechanisms for regulated, policy based cross-border data management. These mechanisms are complemented with final use-case specific AI solution designs.

Solutions for cross-border data management are essential for the functioning of modern economies. On one side, we wish to allow the free flow of data, knowledge and information, and on the other, we also wish to protect company and private information. New Edge-to-Cloud computing technologies such as those integrated by the DECENTER project facilitate the orchestration of sophisticated AI methods across heterogeneous resources which may be widely geographically distributed. Hence, our project provides some interesting scenarios as well as technologies that may be used to facilitate control on the way how data, knowledge and information is stored, processed, and communicated across administrative boundaries.

Within DECENTER, the cross-border data management scenario assumes that a Big Data pipeline may start at one service provider (e.g., camera owned by the municipality of Trento, Italy) and can proceed through secure (e.g., encrypted) Internet channels towards the other processing application stages, which may be implemented in other administrative domains, for example, private or public Cloud providers in Slovenia, and extended further to Cloud providers in Seoul, Korea. Hence, the term “border” in the context of our work refers to any administrative, organisational policy or government regulation in which data should pass. As a result, the process must abide to various specific policies and regulations, requirements for certification, permissions, including personal permissions and preferences.

Requirements extracted by specific parties taking part in a data management scenario may be too hard in which case it may prove impossible to establish the required quality of cross-border data management and transport. For example, regulations differ between countries by requiring collaboration with different cloud providers, implementation of different hardware resources, implementation of different security protocols, empowering people with different rights and similar. Thus, the cross-border data management mechanisms must be designed in a way to allow strict assessment and application of the user's preferences in the specific context where the use of AI will be required.

The goal of this work is therefore to present the outcome of the analysis, design and deployment of specific cross-border data management mechanisms that enable the participating entities control all aspects of the data transport and management when it comes to their administrative domains.

Therefore, in Section 2 we present the specific background in terms of technologies and approaches that may be used to achieve cross-border data management. Conventional data management systems rely on centralized architectures to manage data and suffer from lack of transparency, lack of trust and single point of failure issues. We address these challenges by applying various blockchain technologies with different properties (e.g. immutability, transparency, traceability, safety and security), and programmatic constructs such as Smart Contracts and Decentralised Oracles (also called Smart Oracles). Our work focuses on one specific type of data, which are AI models that are also a topic of research in the DECENTER project. The mechanisms analysed and introduced in the project are initially related to the management of AI based containers that can contain biometric data and in future may be exchanged by countries. However, such artifacts may be subjected to regulations (e.g. GDPR) when it comes to their cross-border orchestration.
Section 3 focuses on the definition of trust in the context when our data passes across administrative borders. In order to analyse the concept of trust in this specific context we introduce a new model for trust management in AI-based Fog Computing applications around a specific new model of trust. Relevant trust attributes are studied in detail.

Section 4 presents an innovative regulatory mechanism which relies on multi-party Service Level Agreements which are prepared as Smart Contracts. This is a relevant aspect of the cross-border data management, because it determines the terms of service and use in the context of usage regulations, expected Quality of Service (QoS) of the AI-models as formal agreements between the involved entities. Hence, the complete life cycle of the AI-models within the cross-border data management scenario is covered.

Further to this setup which sits on top of our DECENTER system architecture, we have introduced a specific system for AI Model (Data) Management that manages AI model distribution onto deployed AI microservices. By combined with the Smart Contracts, it can provide methods to guarantee access to a specific AI model from deployed AI microservices. Section 5 presents its architecture and explains the details of its operation and section 6 presents the implementation of the AI management scenario.

The AI models are stored in a newly designed repository. Section 7 presents the design, implementation, and integration of the repository. Plenty of technical details are provided including the AI model repository design, the semantics of the main elements, the implementation environment, the REST API interfaces for AI model access, implementation results and so on.

AI model repository provides methods for exchanging AI models on running AI microservices, which can be very beneficial to the Federated Learning process. Hence, FL is also supported in the context.

Additional aspects covered by the deliverable concern guidelines for building AI application with DECENTER presented in Section 8. In addition, Kubeflow and DECENTER Model Serving approaches are presented in Section 9. Our current aim is to improve the presentation of our results and prepare a publication that will disseminate the lessons learnt from our work on cross-border data management scenarios.

2 Background

2.1 Blockchain, Smart Contracts and Smart Oracles

2.1.1 Blockchain

Our cross-border AI model management scenario relies on the use of blockchain. In the following we explain some of the blockchain essentials.

Blockchain is a linked-list data structure that forms an ordered list of reverse-linked transactions blocks. A block contains the cryptographic hash of its previous block, the transaction data itself and a timestamp of the transaction. Every block is therefore linked to the previous one, which makes changing data in the blockchain infeasible. The structure is designed to withstand network attacks and can be stored in files or a simple database [1].

Blockchain technology delivered a new form of business or economy because it allows the elimination of the intermediate and exchange transactions in a decentralised way; thus, it
enables global scale, transparent, auditable, and secure applications [2], [3]. Decentralisation refers to the decisions taken by the network (user group) to maintain its consistency, so to make a change in the network a consensus must be reached among the participants in the network. Hence the decision for data change cannot be monopolised by a certain individual or group. As a result, there are various requirements for a consensus mechanism, such as:

- Agreement: All trusted nodes in the network decide on the same value.
- Integrity: A node can make the decision only once in a single consensus cycle.
- Validity: The value agreed upon by all nodes must be the same as the initial value proposed by at least one trusted node.
- Fault tolerant: The consensus algorithm should be able to run correctly in the presence of faulty or malicious nodes (Byzantine nodes).
- Termination: All honest nodes terminate the execution of the consensus process and eventually reach a decision.

To facilitate final state of agreement in distributed systems (i.e., blockchain networks), a consensus mechanism must be implemented. In other words, consensus is a concept that is used in blockchain to provide means of agreeing to a single truth by all entities in the blockchain network. In the following are the most common consensus algorithms:

- **Proof of Work (PoW):** relies on the proof that sufficient computational resources have been spent before proposing a value for acceptance by the network. PoW is used in Bitcoin, Litecoin, Ethereum 1.0 and other cryptocurrency blockchains.
- **Proof of Stake (PoS):** relies on the idea that a node has an adequate stake in the system, meaning that the node has invested enough in the system and any malicious attempt by that node would not outweigh the benefits of staking in the network. PoS is used in DASH, Ethereum 2.0, NEO, Tezos and other cryptocurrency blockchains.
- **Delegated Proof of Stake (DPoS):** is an innovation over standard PoS, whereby each node that has a stake in the system can delegate the validation of a transaction to other nodes by voting. DPoS is used in Lisk, EOS and other cryptocurrency blockchains.
- **Proof of Authority (PoA):** utilizes the identity of the participants, also known as validators, as a stake on the network. They are known in the network and have the authority to propose and validate new blocks as defined in the blockchain rules. Although PoA has very advanced work models, it is not widely used in practice. This consensus algorithm is most applicable to private blockchains because there are no disagreements within the system.

The need for decentralised applications, in domains such as: asset management, healthcare, data management, identity management and so on, is emerging by the amount of data generated by different services. Moreover, recent history of leaking information by service providers, encourages the implementation of decentralised application and the distribution of data in a decentralised manner.

The block is the fundamental structure of the chain. It stores the transactions and is divided into three parts: a header, a transaction counter, and a transaction. The header has three metadata blocks: the first refers to the hash of the previous block; the second adds the data from the timestamp, difficulty and nonce; and the last block is the root of the Merkle tree. A server timestamp adds the date and time to a specific data at the present time, which was created by a computer, making sure that data existed at a certain point of time. To produce
this, it is necessary for an authority called Time Stamp Authority (TSA) to publish and verify
the timestamp. However, before this process, the data must be encrypted using a hash
function [4]. In the case of Blockchain, it serves as a certifying of the creation date for data,
with no need of a central authority. Blockchain also provides integrity to the data because each
block has the previous timestamp in its hash, thus forming a chain that makes it difficult to
change the timestamp. After confirming the timestamp by the network, the difficulty to modify
it progressively becomes almost impossible as the chain becomes longer, thus increasing the
confidentiality of the data [5].

Hash functions refer to a family of non-injective functions that compress certain data of any
size (for example a string, binary files, or TCP packages) into a string of a fixed size. They are
essentially one-directional functions, meaning that while it is simple to compute a hash of some
value, calculation of one of its inverse values is computationally impractical. It is therefore
difficult to restore the original value from its hash. Therefore, hash functions can be used in
applications that require privacy, authenticity, integrity, and non-repudiation of the carried
information.

Another advantage of using hash functions is the reduction of storage space, if the application
allows to store hashes instead of the original data, given that the original data is larger than
its corresponding hash value [6]. Among various applications forms hash functions can
highlight password protection, digital signatures construction, building blocks in authentication
protocols, and cryptographic algorithms structuring [7].

Blockchain uses a variant of the Secure Hash Algorithm (SHA), namely SHA-256, for
calculation of hashes to verify the transactions and calculate the PoW. SHA-256 is a hash
type that turns smaller two 64-bit messages into a hash of 256 bits. Therefore, SHA-256 uses
the iterative structure of Merkle-Damgard to transform a resistant collision and fixed-size
compression function into a hash function that accepts values of any size [8].

Despite offering protection against network attacks, the blockchain technology is not immune
to vulnerabilities and may suffer attacks that compromise the entire security aspect that is
proposed. The most common attacks on Blockchain are the majority hash rate attack (i.e.,
51% attack) and double spending. Although some security vulnerabilities are recognised, they
are mitigated by design, for instance by requiring six confirmations by the network for a
transaction, or by preventing a user and/or user group, having considered the amount of
computer control. Therefore, improved system security in the scope of the cross-border data
management (i.e. blockchain-controlled operations for data management), is considered by
default, due to the security improvements that are delivered by default from applying the
Ethereum [9] and ChainLink [10] technologies and following their recommended security

2.1.2 Smart Contracts

Smart Contracts as a concept were initially defined in a study called “Formalizing and Securing
Relationships on Public Networks”, around 20 years before the development of Bitcoin [12].
In that study, smart contracts were defined as follows: A smart contract is an electronic
transaction protocol that executes the terms of a contract. The general objectives are to satisfy
common contractual conditions (such as payment terms, liens, confidentiality and even
enforcement), minimize exceptions both malicious and accidental, and minimize the need for
trusted intermediaries. Related economic goals include lowering fraud loss, arbitrations and
enforcement costs, and other transaction costs.
Bitcoin first implemented initial form of smart contracts (i.e., scripts) that allowed transfer of bitcoins between entities, although their functionality is rather limited compared to its competitors, such as: Ethereum or Tezos. Apart of P2P payments these solutions also allow their smart contracts to facilitate applications’ business logic. In the context of blockchain, smart contracts can be defined as follows: A smart contract is a secure and unstoppable computer program representing an agreement that is automatically executable and enforceable [13].

Although, smart contracts are named smart, they have deterministic nature, and they can only execute what have been instructed to do and produce the same output each time they are executed. Having in mind the consistency requirements that blockchain networks have, the deterministic nature of smart contracts is highly necessary.

2.1.3 Smart Oracles

Using blockchain and SCs within existing Cloud architectures has much potential. Zhang et al. [14] presented TOWN CRIER (TC) aiming to provide trustworthy (trustful) data to SCs through a middleman service (TC Server). Smart Oracles are useful means that reduce the necessity of costly operations on a blockchain, such as storing and using data within SCs. In particular, Smart Oracles (i.e., Blockchain Oracle) are services that provide the SCs with off-chain data. Specifically, external data provided by Smart Oracles can be used within an SC to decide, if a Fog node can be trusted, and consequently used to deploy an AI container on the Fog node automatically. Advanced Smart Oracle solutions, such as Oraclize\(^1\) provide Smart Contract templates, which ensure Oracle correct data flow. Another Smart Oracle solution is the Ethereum based Chainlink network\(^2\) that provides reliable tamper-proof inputs and outputs for SCs on any blockchain. These few useful studies form the basis for the present work, which aims at using blockchain, SCs and Smart Oracles to provide regulatory and policy enforcement aspects, alongside with transparency, traceability, and a great level of autonomy to the DECENTER’s Fog Computing Platform.

2.2 AI Model Management and Related Techniques

2.2.1 Libraries and repositories for easy use of AI models

In recent years, libraries, and tools for generating and managing AI models have been actively developed and introduced. There are high demands of reusability of AI models from industry since it takes a lot of resources and time for training of an AI model. Many companies are providing model zoo or repository to provide its pre-trained models and to make it easy to retrain it. In DECENTER, we’re focusing on building service on an architecture with loosely coupled microservices, and this AI model reusability is of importance. This chapter introduces tools for creating and reusing AI models.

2.2.1.1 Libraries

The following introduces representative AI libraries used when creating AI models.

- **TensorFlow** is an open-source library to create machine learning models for desktop, mobile, web, and cloud. TensorFlow is a very powerful and mature deep learning

---

\(^1\) [http://www.oraclize.it/](http://www.oraclize.it/).

\(^2\) [https://chain.link/](https://chain.link/).
library with strong visualization capabilities and several options to use for high-level model development. The library works with basic datatype ‘tensor’ that is a multi-dimensional array. Tensorflow1.0’s model is composed static graph which after adding node and edge to the graph and then the calculation is implemented on session. In other hands, Tensorflow 2.0 offers mode flexible model that is implemented with defining code at the same time. Because there are many documentations and a huge community, developers can find reusable code in various case.

- **Pytorch** is a python open-source library that performs immediate execution of dynamic tensor computations with automatic differentiation and GPU acceleration. It defines dynamic graph which is a method named ‘define by run’ that assigns value while creating graph simultaneously and is more Python friendly than TensorFlow. So, user can write code clearly and intuitive. With these features, many research papers use Pytorch for modeling.

- **Keras** is a deep learning API written in Python, running on top of the machine learning platform TensorFlow. Keras does not do its own low-level operations, such as tensor products and convolutions. it relies on a back-end engine for that. Thus, it supports multiple back-end neural network computation engines. However, its primary (and default) back end is TensorFlow, and its primary supporter is Google. The Keras API comes packaged in TensorFlow as ‘tf.keras’. The code is developed user friendly and readable so that developer can architect neural network model using just a few lines.

- **CNTK (Microsoft Cognitive Toolkit)** is a unified deep learning toolkit that describes neural networks as a series of computational steps via a directed graph. In this directed graph, leaf nodes represent input values or network parameters, while other nodes represent matrix operations upon their inputs. CNTK supports feed-forward, convolutional, and recurrent networks for speech, image, and text workloads, also in combination.

- **Scikit-Learn** is a python machine learning library which was initially developed by David Cournapeau as a Google summer of code project in 2007. Scikit-learn provides both of supervised and unsupervised learning algorithms via a consistent interface in python. The library contains a lot of efficient tools for machine learning and statistical modeling including classification, regression, clustering, and dimensionality reduction. These a lot of tools for machine learning algorithms is one of the big reasons for the high usage of scikit-learn.

2.2.1.2 Repositories

The following introduces representative AI model repositories used when reusing pre-defined AI models.

- **Tensorflow Hub**\(^3\) is a repository of reusable assets for machine learning, which is a platform to publish, discover, and reuse parts of machine learning modules in TensorFlow. It makes easier for developers to share, reuse, and debug machine

\(^3\) [https://tfhub.dev/](https://tfhub.dev/)
learning models. TensorFlow Hub was launched in March 2018 as a repository that hosts pre-trained machine learning models developed by Google and the AI company DeepMind. The repository allows users to search among more than 1,000 models using different criteria such as domain (image, text, or video) or model format (TF.js, TFLite or Coral). In particular, it provides pre-trained SavedModels that can be reused to solve new tasks with less training time and less training data. The Tensorflow hub Python library supports to download and reuse SavedModels in TensorFlow program with a minimum amount of code.

Figure 1. Training and deployment environment of Tensorflow [15]

- **TorchServe** is an open-source model serving framework for PyTorch that makes it easy to deploy trained PyTorch models permanently at scale without having to write custom code. It provides default handlers for the most common applications such as object detection and text classification, so developers do not have to write custom code to deploy their models. TorchServe offers multi-model serving, model versioning for A/B testing, metrics for monitoring, and RESTful endpoints for application integration. TorchServe takes a pytorch deep learning model and wraps it in a set of REST APIs. Currently it comes with a built-in web server that user run from command line. This command line call takes in the single or multiple models user want to serve, along with additional optional parameters controlling the port, host, and logging. TorchServe supports any machine learning environment, including Amazon SageMaker, Kubernetes, Amazon EKS, and Amazon EC2.

Figure 2. Torch Serve Process [16]

- **Google AI Hub** offers a collection of assets for developers and data scientists building artificial intelligence (AI) systems. Al Hub is also platform that allows data scientists and developers to store components such as pipelines, Jupiter Notebook, and
TensorFlow modules in a single, secure place for collaboration. Users can find or deploy ML pipelines and explore and reuse TensorFlow modules. AI Hub holds user’s content from many places: Kubeflow Pipelines, Cloud Video Intelligence, Cloud Translation, Cloud Vision, Kaggle, Cloud AutoML, Cloud Text-to-Speech, Cloud Speech-to-Text.

- GitHub is a website and cloud-based service that helps developers store and manage their code, as well as track and control changes to their code. The repository can be public or private. Many developers can also upload and download their machine learning model on the repository.

2.2.2 AI model as Data

An AI service generates outputs by applying inputs to its model, and in this manner, it can be said that AI service consists of three kinds of data: AI model, input, and outputs of AI model. AI model is trained with data for its purpose and can be improved with further training with new data. When an AI model is updated or improved, this data (the AI model) can be substituted in an AI microservice to provide better service performance. Further, this AI model can be related to the cross-border restrictions. If an AI model is to identify a person from incoming video stream, it needs to check the compliance with local regulations such as GDPR or PIPA. To this end, it is important to have methods to manage AI models as data in systematic way. In this section, the characteristics of AI model and its related protocols are presented.

Artificial intelligence is anything which enables devices to have a brain like a human. The device can have intelligence by receiving a proper AI model. In terms of AI applications, application-related data must be transferred from the cloud to the target device to maintain reliable service. Cloud must manage AI-application-related data efficiently and deliver proper data to the device. In that case, the device that receives the related data including AI model can do a specific task such as decision making, classification, and prediction.

There are works on AI model deployment on the field, however those works are focusing no AI models onto cloud resources rather than edge resource, which are usually have very limited resources. Therefore, it is not easy to explicitly manage, register and utilize modified models according to the service characteristics and purposes. In this project, micro-service is considered as an independent unit of monolithic service in allocated resources. Therefore, the AI model repository is designed to distribute AI models in various units of micro-service.

We suggest the AI repository server which is able to distribute AI models to each microservice. Stored AI models can be easily distributed to microservices by connection between AI model repository and each micro-service. For example, if the resource on which micro-service is installed has relatively poor performance, it may be better to choose a lightweight AI model to use, even if the accuracy is degraded. We take this requirement for the future scenario and designed a repository server deploying AI models efficiently.

2.2.2.1 Formats and Serialization Methods of AI Models

A trained AI model is serialized and stored in a file for future use. For example, a store AI model file can be transmitted to another device (such as Fog node) to be used in an AI application, or can be loaded onto memory for re-training. Although the needs of a standard format for the AI model serialization are high, there are multiple serialization formats being used in the field. Several famous formats for AI serialization are described in this section.
2.2.2.2 **Hierarchical Data Format**

Hierarchical Data Format version 5 (HDF5) [17] is a data format and a library designed for efficiently storing and accessing large amounts of heterogeneous data. It was initially developed as a tool to store large amounts of complex data and is mainly used for storage of large-scale image data, flight recordings of aircraft and ships. It supports an unlimited variety of data types and is very flexible for handling efficient I/O and complex data. The HDF5 library can be seen as a kind of an embedded database that does not provide a database management system itself, unlike some other systems. When used with Keras, the high-level artificial neural network framework, the weights of a model can be stored into the HDF5 format while the model architecture can be saved using JSON or YAML format.

2.2.2.3 **Protocol Buffer**

Protocol buffer[4] is a protocol developed by Google and used for serialising structured data. It is extensible, and, unlike the pickle module, programming language- and platform-agnostic. Thus, protocol buffers can be written in various programming languages, including Java, C++, and Python. When used with the TensorFlow artificial neural network framework, both the graph definition and the weights of a model can be represented as a protobuf (.pb) file.

2.2.2.4 **Open Neural Network Exchange Format**

The Open Neural Network Exchange Format (ONNX)[5] is a new open standard format for representing an AI model. The ONNX enables AI models to be trained in a specific framework, such as CNTK or TensorFlow, and to be later reused in another framework for inference. ONNX models are therefore interoperable and supported in common famous frameworks such as Caffe2, CNTK, TensorFlow, Core ML, MXNet, and PyTorch. The ONNX is based on protobuf in which strings can identify types of elements in the model’s graph. A predefined dictionary of operators and definitions is provided. Model can be saved in the model.onnx format to use ONNX for interoperability. This format is serialised representation of the model in a protobuf file.

2.2.2.5 **Neural Network Exchange Format**

Like ONNX, the Neural Network Exchange Format (NNEF)[6] is an exchange format to execute networks trained with different frameworks on different platforms. The NNEF is intended to describe the network structure and the trained parameters of the model and defines a container that contains a network structure and a set of tensor data files. The network structure includes used operations and activation functions. The goal of the NNEF is to enable a developer who made an AI model to easily transfer the trained networks into a wide variety of inference engines. This format can find its use when the intelligence is required on a variety of edge devices which makes an AI model widely available.

2.2.2.6 **Pickle**

The Pickle[7] module implements binary protocols for serialising and deserializing Python object structures. As the serialisation uses Python-specific data format that is suitable for representation of common Python objects, the de-serialisation is typically done exclusively in

---

4 https://developers.google.com/protocol-buffers
5 https://onnx.ai/
6 https://www.khronos.org/nnef
7 https://docs.python.org/3/library/pickle.html
Python and not in some other programming language. In practice, the Pickle module has various use cases; one notable AI library that utilises it is the Scikit-learn package.

Having the baseline technologies presented, we can now better understand the information for cross-border data management and AI model management that will be presented in the following sections.

3 Trust Management for AI Based Fog Computing Applications

This section provides final research results of trust management in the context of cross-border data management. Essentially, we have researched trust management and trust attributes in the context of cross-border data management for the purposes of establishing dependable and trustworthy interactions between the entities, stakeholders and services in a cross-border scenario. While trust, as we already defined, is a binary concept, our approach towards data management in the fog should be soft in the sense that one must always balance between the various trade-offs. For example, if one AI model for person identification that must be transferred from Korea to Europe contains sensitive information, in such case more regulations and policies (e.g., GDPR, PIPA) would apply, on another occasion, perhaps only the time of the transfer of the model (as fast as possible) would represent the trust-issue.

This section complements the initial research results that were described in D4.2 [18]. In particular, the definition of trust and the list of important trust attributes was further investigated, and the findings are described in this section.

3.1 Definition of trust in smart environments

The concept of trust is complex similarly to many aspects of human endeavour. A suitable definition of trust was presented by Gambetta: trust (or, symmetrically, distrust) is a particular level of the subjective probability with which an agent assesses that another agent or group of agents will perform a particular action, both before he can monitor such action (or independently of his capacity ever to be able to monitor it) and in a context in which it affects his own action.

So far, trust has been studied in different fields and domain, such as social sciences, economics, philosophy, and cyberspace. Consequently, trust has been defined differently in the literature, depending on the views of the authors and on the context of trust. The most notable definitions on trust are as follow:

- Definition 1: Mayer et al. [19] defined trust as the willingness of a party to be vulnerable to the action of another party based on the expectation that the other will perform a particular action important to the trustor, irrespective to the ability to monitor or control that other party.
- Definition 2: In the context of online transactions, Kimery et al. [20] defined trust such as, online trust is a customer’s willingness and enables to accept an online transaction according to their positive and negative expectations on future online shopping behaviour.
- Definition 3: Corritore et al. [21] defined online trust as "an attitude of confident expectation in an online situation of risk that one’s vulnerabilities will not be exploited".
Definition 4: Chang et al. [22] defined trust as the belief that the trusting agent has in the trusted agent’s willingness and capability to deliver a quality of service in a given context and in a given timeslot.

Definition 5: Buttyan et al. [23]: Trust is about the ability to predict the behaviour of another Party.

Definition 6: Aljazzaf et al. [24] defined trust as follows: "Trust is the willingness of the trustor to rely on a trustee to do what is promised in a given context, irrespective of the ability to monitor or control the trustee, and even though negative consequences may occur".

Definition 7: Daubert et al. [25] defined trust in the context of IoT as device trust, entity trust, and data trust, where trusted computing and computational trust could be used to establish device trust. Entity trust refers to the expected behaviour of participants such as persons or services. And trusted data may be derived from untrusted sources by aggregation or may be created from IoT services where data require trust assessment.

In computer science, there have been efforts to formalise trust, for example, to facilitate cooperation among autonomous agents [26]. The formalisation of trust, therefore, aims at improving the possibilities for trust management in dynamic and distributed computing environments. In this context, Viljanen presents various considerations towards the definition of an ontology of trust [27].

In summary, the trust definitions include one or several attributes (e.g. dependency, confidence expectation, vulnerability, reliability, comfort, utility, context-specificity, risk attitude, or lack of control) that often are perceived as attributes related to the QoS. Accordingly, there is not standard definition of trust; nevertheless, it is obvious that the main goal of trust management is leveraging security by assisting in decision-making processes.

### 3.2 Relevant trust attributes

The goal of this section is to analyse key aspects and attributes to trust that are important for smart applications and environments. Trust management requirements depend on the specific use cases where trust management must be applied and are multifaceted. The fundamental trust attributes are as follow:

- **Availability** is a fundamental attribute of Fog nodes that evaluates the probability of the node’s correct functioning at a specific moment in time.

- **Credibility** defines the degree to which the data source or the data is seen to be believable, a concept that can be extended to any data item, such as a video frame or an AI model based on TensorFlow.

- **Privacy** in the context of the IoT includes the following relevant aspects: awareness of security risks imposed by smart devices surrounding a human subject, individual control over the data collection and processing of personal data, and awareness and control of subsequent use and dissemination of personal information by those entities to any entity outside the subject’s personal control sphere.

- **Response time** is an attribute that represents the time necessary for data to be processed by a selected Fog node and data packets to be transferred to the client. For simplicity, in the present implementation, it is measured as the round-trip time for the
package to reach a microservice, perform necessary processing, and return extracted metadata to the client.

- **Throughput** is an attribute that represents the rate at which data is transferred between endpoints (e.g. between a sensor and a Fog node). In other words, this attribute shows the amount of data that a specific component can successfully transfer per unit of time.

- **Security** estimates the ability to protect the system from accidental or intentional external attacks. This attribute to trust is closely related to confidentiality that evaluates if the data within the distributed environment is protected from disclosure to unauthorised entities.

- **Transparency** is an attribute that allows the blockchain ledger to be fully auditable. That allows everyone participating in the ecosystem to view the stored transactions on the blockchain.

- **Traceability** is an attribute that is closely related to transparency. Traceability allows to trace back the interaction between entities on the blockchain since all history can be traced back to the first transaction.

However, there is a large choice of trust attributes that can be derived by combining multiple fundamental attributes into one, as listed below:

- **Trust Relationship and Decision (TRD)** is an attribute that provides an effective way to evaluate trust relationships among entities in smart environments and assist them to communicate and collaborate with each other. The trust relationship is not absolute. In other word, the purpose of trust, the environment of trust (e.g., time and location), the role of the evolved actors, and the risk of trust are defined a priori. For example, a trustor can trust a trustee to forward a data packet in one context; however, the same trustor cannot trust a trustee to do another task in another context.

- **Data Perception Trust (DPT):** is an attribute that ensures reliability and trustworthiness of data accumulated by sensors in smart environments. In this context, DPT covers sensor sensibility, preciseness, security, reliability, and persistence, as well as data collection efficiency.

- **Data Fusion and Mining Trust (DFMT):** is an attribute related to ensuring trusted social computing based on users' social behaviors and social relationship analysis, in the context of reliability, holographic data process, privacy preservation and accuracy. DFMT concerns the objective properties of the data processor in the IoT network layer. Current advance of DFMT have not yet been applied in practice.

- **Data Transmission and Communication Trust (DTCT):** is an attribute related to IoT security and privacy that requires light security/trust/privacy solution. This attribute assures that the sensed and processed data will be transmitted and communicated securely in a trustworthy way.

- **Quality of IoT services (QIoTS):** is an attribute related to the quality of personalized IoT services should be ensured whilst maintaining a high level of security. This attribute implies that IoT services should be personalized and offer their services in the right time, in the right place and to the required entity.

- **Identity Trust (IT):** is an attribute that assures trust based on entities’ identity. This attribute usually is satisfied through the implementation of various types of identity management mechanisms, reputation mechanisms and similar. It concerns the objective properties of the system (e.g., identity privacy) and subjective properties of the participating entities (e.g., user hope, user reputation) and relationship context and activities that may influence identity management policies.
4 Multi-Party Service Level Agreements for Multi-Party Data Governance

This section is devoted to the latest trends in software engineering, which are based on building trustworthy, flexible, and reusable AI applications by implementing a multi-tier application architecture. Splitting AI methods into several computing tiers that may belong to different administrative domains, such as into Front and Rear parts of Deep Learning Neural Networks, may lead to greater privacy and security of the information when it is being executed in the Fog [28]. Currently, the key problem is that the Edge-to-Cloud continuum offers great heterogeneity of possible deployment options coupled with dynamically changing operational conditions and location regulations. This can be addressed by tailoring Service-Level Agreements (SLAs) for the Edge-to-Cloud continuum that implement blockchain-based mechanisms such as Smart Contracts and trustless Smart Oracles and can be used to implement mechanisms for the dynamic federation of computing resources coupled with transparent and traceable orchestration.

4.1 Architecture for multi-party SLA management

In the following, we elaborate our novel SLA management architecture and system implementation that is designed to provide high QoS operation to DECENTER's smart applications. The goal of our proposed SLA management architecture is to facilitate an automated and transparent decision-making process for (re)deployment of multi-tier applications in the Edge-to-Cloud computing continuum.

The proposed architecture implements MDP method [29] that aids to automatically rank the available deployment options according to prior usage information, current monitoring data and QoS requirements that are precisely defined within the SLA.

Figure 3. Architecture for trusted multi-party SLA management

To achieve these technical goals, the proposed architecture allows registering available deployment options by providers, definition of SLA user requirements and autonomous deployment and redeployment of applications among the available deployment options. The architecture can be observed through three scenarios, which are the following:

- registering a certified deployment option,
- automated deployment of applications and
• automated redeployment of applications.

These three scenarios are elaborated in the following subsections.

The high-level architecture design for trustworthy SLA management that implements SCs is depicted on Figure 3. Each architecture level is described in the following.

• **Application Layer** is an application with an intuitive graphical user interface (GUI). It is an entry point that the software engineer uses to define QoS requirements, which are incorporated into SLA. This layer is directly communicating with the Blockchain Layer, which is the Ethereum ecosystem. Thus, it allows the user to trigger a SC execution. For the Application Layer to communicate with the Ethereum (ETH) ecosystem, it implements ETH bridge called Metamask\(^9\), which plays pivotal role in the process. Primarily, Metamask is an Ethereum wallet that allows users to: (1) create and switch accounts that can be used in various ETH networks (e.g., Main ETH network, Ropsten, Kovan or Rinkeby); (2) perform transactions between accounts. It facilitates the interaction with the Ethereum ecosystem by injecting a Javascript library called web3.js \(^{30}\).

• **Blockchain Layer** is necessary to automate the SLA management process and empower fairness between the involved parties (i.e., service providers and consumers) and executes traceable and transparent transactions on the Blockchain. The blockchain implementation is based on public Ethereum ledger as a public blockchain environment, because it is composed of two main components: SC templates and Smart Oracles. There are two main types of SC utilised in the system: SCs for registration of deployment options on the blockchain and SCs for automated (re)deployment of applications. The deployment of the SCs occurs on demand through the blockchain service which plays the role of a non-biased system that executes the SC and pays the service provider in case the SLA is not violated. However, in case there is an SLA violation, the SC terminates, pays the service provider for the service provided until the moment of the SLA violation, whilst compensating the service consumer for the remaining of time. A more detailed overview of the SCs work and the SLA management workflow is elaborated in the sections bellow.

However, SCs by default cannot act outside the blockchain, thus they are not capable to retrieve off-chain data. Since SCs in this SLA management system must communicate with external services, such as computing nodes, QoS monitoring system or decision-making mechanisms, Smart Oracles had to be implemented. The Smart Oracles are trusted third-party services that provide means for SCs to communicate with registered APIs from the external services. This approach results in enhanced integrity of the functions that verify the correctness of the API queries by using unique API keys and thus avoid calls from potential malicious SCs.

• **Decision-Making Layer** estimates the optimal deployment option for deployment of containerised software components and initiates the container deployment process. To estimate an optimal deployment option, this layer queries a Smart Oracle from the Blockchain Layer to retrieve monitoring data and prior usage knowledge only for

---

\(^9\) [https://metamask.io/]
deployment options, which are registered on the blockchain. This layer is composed of components that are products of our earlier research work and they have been thoroughly described in D3.4 [29].

- **Edge-to-Cloud Computing Layer** is composed of deployment options, which are registered on the blockchain by the service providers. The deployment options are used for deployment of the containerised two-tier applications, where each tier of deployment options play a different role. For instance, the Edge-based deployment options are responsible for running the software components that require less computing power, whereas the other components run on the Cloud/Fog-based deployment options.

### 4.2 Experimental evaluation

The Blockchain Layer and its SCs are crucial in the process of achieving automated and trustworthy operations. Hence, they have an important role in the system architecture. As a result, the individual functions including the SC deployment must be efficient in terms of cost and execution performance. In comparison to the traditional SCs interacting fully with on-chain data, our SCs “A” and “B” presented in Figure 3 include Smart Oracle mechanisms that increase the data interaction on an off-chain level but at the same time slightly increase the overall cost of the Smart Oracle enabled functions.

![Figure 4. Gas consumption of all SC functions in SCs “A” and “B”](image)

In our experimental environment we used the Ethereum testing environment Rinkeby and we presented the averaged results of 10 executions, running own Chainlink nodes with the determined Smart Oracle cost per interaction of 0.001 LINK tokens, which is equivalent to the main Ethereum environment of approximate 0.037 USD. The evaluated SCs (“A” and “B”) from the gas consumption metric are depicted in Fig.4. The results indicate that the most expensive functions are constructors and following the Smart Oracle enabling functions. Since the deployment of the SC “A” is performed only once and the deployment of the SC “B” once per service consumer, the SCs are costly feasible. Moreover, the majority of the functions is triggered by the Service Provider stakeholders (e.g., Service owner, Cloud providers, etc.), while only 4 functions (triggerSC, payService, checkLockState and triggerStop) are executed from Service Consumer stakeholder. The Service Consumer SC functions are relatively unexpensive and thus make the SCs overall costly acceptable.
It is known that the execution of SCs is possible within one block or multiple ones. We focused on the main three functionalities: registration, deployment, and redeployment. The results shown in Table 1 indicate that the fastest operation is registration, due to the simplicity of the SC functions, but the deployment and redeployment execution time varies from 1 to 2 blocks depends on Ethereum network load and used transaction fee. In the case of low execution time, it is used high transaction fee of $20 \cdot 10^{-9}$ ETH, otherwise in the case of high execution time it is used low transaction fee of $1 \cdot 10^{-9}$ ETH. Even though the performance of the SC functionalities is consuming, and it is not possible to reduce the execution below one block (that is approximate 15 seconds), the trust benefits among the involved entities prevail.

Table 1 Performance analysis for registration, deployment, and redeployment operations

<table>
<thead>
<tr>
<th>Operation</th>
<th>Low execution time [sec]</th>
<th>High execution time [sec]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Registration</td>
<td>12</td>
<td>15</td>
</tr>
<tr>
<td>Deployment</td>
<td>14</td>
<td>24</td>
</tr>
<tr>
<td>Redeployment</td>
<td>13</td>
<td>18</td>
</tr>
</tbody>
</table>

4.3 Summary

The proposed system for trustworthy SLA management was fully tested and evaluated. Our evaluation addressed the performance of the newly implemented blockchain solutions. The blockchain implementation was evaluated in terms of cost and execution performance because these criteria have an essential role if we intend to achieve business value. The source code for the Blockchain layer components (i.e. smart contracts and oracles), is available in DECENTER’s WP4 repository on GitLab\(^\text{10}\).

Our results have shown that the blockchain operations take at average no more than 24 seconds to execute, thus allowing the system to timely address any SLA violations, while at the same time satisfying high security and trust related standards that are offered by this technology. On the other hand, the transaction fees for the operations depend on the amount of traffic that the blockchain is experiencing and the required execution time for each operation. In other words, the faster the operations need to be executed, the more expensive it will be. Although, the transaction fees were significantly low during the evaluation process, there is still space to further optimise the performance of our blockchain implementation. In summary, the proposed SLA management approach can deliver QoS-aware deployment operations, whilst at the same time maintaining trust relationships among the involved entities.

This section covered the challenge on building trustworthy, flexible, and reusable AI applications. However, to achieve trustworthiness in cross-border scenarios also requires careful implementation of regulations and legislations to protect personal data, which will be described in the next section.

\(^{10}\) https://gitlab.fbk.eu/decenter/w4t3
### 5 AI Model (Data) Management Operations and Architecture

This section describes the final scenario of cross-border data management – a Data Management Usage Scenario which is being implemented with our new Data Management Architecture. The scenario addresses a scenario of Korean citizen (John) visiting a construction site in Slovenia, which is a Member State of the European Union that requires a simple, efficient, and rapid identification of the person together with compliance with European and Slovenian legislation regarding the protection of personal data.

#### 5.1 GDPR and PIPA regulations

##### 5.1.1 GDPR

The General Data Protection Regulation (GDPR) is a regulation within the EU law on data protection and privacy for the EU and EEA. This regulation imposes obligations onto organisations which target the data of EU citizens. Its 99 articles regulate the rights of any European citizen whose (personal) data is being accessed, collected, processed, or sold.

The GDPR’s purpose is to protect the individual’s right to their personal data. It does this to comply with the European Convention on Human Rights, which among others also states an individual’s right to respect for private and family life, which also encompasses the individual’s automatically or manually collected personal (digital) data. GDPR is prepared also to create a uniform level of protection and security of personal data throughout the European Union, so that all organisations targeting individuals’ data are upheld to the same standards, and that free data sharing, and movement is uninterrupted.

To enforce: lawfulness, fairness, transparency, purpose limitation, data minimization, accuracy, storage limitation, integrity, confidentiality, and accountability, the GDPR regulation differentiates three roles that are to be enforced in all systems complying with the regulation, namely:

- **Data Subject**: represents an individual whose data is being processed or collected. This can be anyone in the case of public blockchains, or in the case of private or consortium blockchains, certain users who have granted access and certain permissions to their data.

- **Data Controller**: represents a person or an entity that is usually responsible for data access management. Essentially, this entity determines if, how and why the data will be processed.

- **Data Processor**: usually are third party entities that wish to access, collect, and process Data Subject’s data. They process this data on behalf of the Data Controller, who does not have this authority. In some cases, with a given consent from the Data Subject, the data can be sold to the Data Processor, from the Data Controller.

Important articles from the GDPR regulation are as follow:
Rights of the data subject:

- **Article 13: Information to be provided where personal data are collected from the data subject**

  The article defines what information should be provided by the data controller to the data subject, when personal data related to the data subject is collected. In addition, this article also ensures fair and transparent processing of the data during the time period that the data controller has control over data subject’s personal data.

- **Article 15: Right of access by the data subject**

  Defines whether and how information about the processing of personal data is provided to the Data Subject. The personal data processing information is also always transparent, again either with an implemented access policy, strict consent checking or instruct that the Data Subject is the only one allowed to execute CRUD operations on their data as specified in the default policy of the Blockchain, and these rights are unchangeable.

- **Article 17: Right to erasure (‘right to be forgotten’)**

  Defines whether and how the Data Subject is able to make the Data Controller (or whoever is responsible) erase all personal data. The data subject shall have the right to obtain from the controller the erasure of personal data concerning him or her without undue delay and the controller shall have the obligation to erase personal data without undue delay where one of the following grounds applies

- **Article 18: Right to restriction of processing**

  Defines whether the Data Subject can restrict the Data Controller from processing certain data, and how. With fine-grained access control, it is easy to define which restrictions on data processing are to be put into place by the Data Subject, as they are only obliged to consent to the processing of data they wish to be processed.

- **Article 20: Right to data portability**

  Defines whether the Data Subject can request to receive their data to give it to another Data Controller, and how. The second and third solution grant the Data Subject complete control over either the consent to data usage or over all their data, which in turn grants the Data Subject the right to change the Data Controller’s access anytime they wish to do so.

**Data Controller and Processor**

- **Article 25: Data protection by design and by default**

  Defines the technical and organisational obligations applied to the data controller and processor that are necessary to be implemented to assure that only personal data that is necessary for a specific purpose is being processed. This measure ensures that by
default personal data are not made accessible without the individual’s intervention to an indefinite number of natural persons for the period of data storage.

- **Article 30: Records of processing activities**

  Defines the information that each controller and, where applicable, the controller’s representative, shall maintain in the record of processing activities under its responsibility.

### 5.1.2 PIPA

The Personal Information Protection Act (PIPA) is a Korean law to protect the personal information. The most recent amendment of this Act was on February 2020, which was a major update to ensure an adequate level of protection with GDPR. This Act is one of the three major data privacy laws and the other two are the Act on the Promotion of Information and Communications Network Utilization and Information Protection and the Credit Information Use and Protection Act.

Important articles from the PIPA are as follows:

- **Article 2: Definitions of personal data and pseudonymization**

  Defines the scope of personal information and pseudonymization. The data subject and personal information controller are introduced, and the scope of scientific research is defined.

- **Article 3: Principles for Protecting Personal Information**

  The personal information controller shall specify explicitly the purposes for which personal information is processed; and shall collect personal information lawfully and fairly to the minimum extent necessary for such purposes.

  The personal information controller shall process personal information in an appropriate manner necessary for the purposes for which the personal information is processed, and shall not use it beyond such purpose

  The personal information controller shall ensure personal information is accurate, complete, and up to date to the extent necessary in relation to the purposes for which the personal information is processed.

**Processing of Personal Information:**

- **Article 15: Collection and Use of Personal Information**

  The article defines the conditions and circumstances where personal information may be collected, and role of a personal information controller to inform related information.

- **Article 18: Limitation to Out-of-Purpose Use and Provision of Personal Information**
The article defines the scope where the personal information cannot be used, and role of personal information controller. Also, it defines provision of personal information to a third party.

- **Article 21: Destruction of Personal Information**
  
The article defines when and how the personal information should be destroyed by a personal information controller.

**Safeguard of Personal Information:**

- **Article 29: Safeguard of Personal Information**
  
The article defines duty of the safeguards, to take appropriate measures to manage the data and store them safely.

5.2 Data Management Usage Scenario in compliance with GDPR/PIPA

**Setup and requirements**

In the following paragraphs, the Data Management Usage scenario is described through a use case when an EU citizen travels to Korea (see Figure 5).

The biometric model to verify John is stored in a repository in the EU. To protect his privacy, he decides to use the model that is split into two parts. The inference done in the front part of the AI model is computationally more intense to compute and should be deployed in an edge near the site. The inference in the rear part of the AI model is computationally less intense and is computed in a fog infrastructure that is relatively close to the first infrastructure.

At the construction site entrance, a video camera will record John’s face and perform member verification. Therefore, the construction site needs to deploy the AI method which uses John’s biometric model for person identification. While doing so, John wants to make sure that the EU privacy regulations (i.e., GDPR) are respected. To further reduce the possibility of leaking
private information, it is also necessary to deploy the model on certified infrastructure providers that use special-purpose hardware, such as Intel SGX, to process the data in a secure enclave.

**Step-by-step use case description**

- Using application’s GUI, gives consent for his personal data (i.e., AI model composed of his biometric data) to be used for a specific period by secure processing infrastructures after his arrival at the construction site in Korea.

- After the given consent, the application triggers a Smart Contract function that through the EU regulation Smart Oracle verifies the existence of the required data in the EU repositories; and registers the given consent on the Blockchain. The immutable consent log contains information about the data owner, data storage, data processor and the duration of the given consent.

- Upon arrival in Korea, John’s data must be deployed on secure computing infrastructure in Korea. The construction company provides to its application service provider, John’s wallet address to verify his consent on the Blockchain.

- The application service provider uses the DECENTER fog platform to compose the application where the QoS parameters, provided by the construction company, are selected. The application requires specially crafted container for the AI model to be fetched from the AI repositories in EU at runtime.

- The application composer passes over application manifest to the Resource Selector and the Resource Selector estimates a possible deployment from the pool of trusted Fog nodes in Korea, after which it passes over the deployment plan to the Data Management Module.

- Data Management Module grants access based on data accessed through the Smart Oracles: (1) fulfilled EU regulations, (2) fulfilled Korean regulations, (3) permissions of the AI model owner – the physical person whose AI model is to be transferred.

- When selecting the target AI model, the application service provider has to pay the predetermined amount of tokens from its public wallet for the usage of the model. Input parameters for the Smart Contract are: target model repository URI and the target AI model to use.

- Once the transaction is confirmed, the Smart Contract locks the received tokens and allows access to the requested AI model and returns an API key for the retrieval of the AI model.

- The application manifest (in the form of an SLA) is handed to the infrastructure and John’s biometric data will be available to the construction site throughout his stay in Korea.
6 Implementation of the cross-border AI model (data) management scenario

6.1 Architecture and implementation of AI Model (Data) Governance and Management

The initial architecture for data management module, which is part of the DECENTER Fog and Brokerage platform, was presented in D4.2. The module comprises five dedicated components, which are on the figure designated with blue colour: AI Model Repository, Trusted Model Manager, Blockchain Service, Data Management Service and Smart Oracle. They are explained in the following into more details.

Figure 6 Architecture for Data Management and Governance

After defining the structure and interaction, we have been investigated how data (AI model) can be managed for the cloud-native AI application which are deployed on the distributed infrastructure with DECENTER. As depicted in the usage scenario, the resources for the cloud-native AI applications are going to be selected from various infrastructures and data usage needs to be validated according to the different situations where the resources are placed. Until M30, tasks from WP4 have been related to tasks from WP2 and WP3, in order to improve the existing architecture for the data management module, define clear structure of the Platform and integrate its components.

The figure above describes the initially proposed data management architecture of DECENTER. In the previous architecture, some of the components of data management module are explained only in high-level architecture. For example, Trusted Model Manager exists on both Data Management module and in DECENTER Platform, while AI Model Repository exists only in the Data Management module. This mixed structure of data management components is complex to configure, integrate and run with the DECENTER platform. Following the “Clean code” and “Clean architecture” principles, in this deliverable we
have investigated how to make the structure independent from each other to make the interaction more straightforward and maintenance easy.

The updated structure is reflected in the revised DECENTER Platform, which is described in D2.2. In the revised architecture, the data management exists as a stand-alone module which is independent of the Platform, which gives a clear distinction between the role of each component – the Platform is focussing on optimal resource orchestration, while data management provides one of Applications Service on top of those orchestrated resources.

The data management takes places in *Data Management* module of *Application Service* in DECENTER Platform, and it interacts with two other components – AI Package and Orchestrator. The figure below depicts revised view of the architecture and for data governance and management. Data Management provides the following functionalities with DECENTER Platform as an Application Service.

- **Analysis of cloud-native AI application resources**
  
  It will read the deployment configurations from the Platform, more specifically, the *Orchestrator* component and analyze them. Basically, the data access policy is managed with the Smart Contract, and if the Smart Contract needs to be updated according to the configuration, it will invoke Smart Oracle to reflect off-chain data to update it.

- **Access Off-chain data to update Smart Contract**
  
  If the Smart Contract for the data management needs to be updated, this Data Management will read off-chain data and update it. Here DECENTER makes use of Smart Oracle to reflect dynamic nature of off-chain data to the data management.

- **Grant Access to the deployed microservices**
  
  If everything meets the condition written in the updated Smart Contract, the data management module access grants to the deployed microservices of cloud-native AI applications. Here it makes use of AI Model Repository component, to make it able for a microservice to retrieve AI model of interest without needs of re-deploying AI microservice.

![Diagram of Data Management interaction with other Platform Components](image)

The source code related to the activities in T4.3 is available at DECENTER's GitLab repository.

7 AI Model Repository final design, implementation, and integration

7.1 AI model repository

In DECENTER, we are focusing on delivering AI from the cloud to the edge in efficient ways, instead of building AI for the edge from the scratch. To this end, it is required to handle the resources on the edge in efficient way as well. Unlike the resources on the cloud, where it is considered to have infinite resources, the resourced on the edge are limited according to the hardware they are relying to. Suppose that we have two edge devices for AI, one with the Nvidia 1080ti GPU and the other one with Nvidia Tegra GPU. Even though we have a good AI model with higher accuracy, it is hardly expected to be run on the second edge due to the hardware performance restrictions. To handle those different resources on the edge, a proper model according to the resource capabilities needs to be deployed. To provide AI models efficiently, DECENTER took approach with separating AI model delivery from the AI microservice container. Instead of storing AI model in a container for deployment, a container without AI model will be deployed onto resources, and AI model will be deployed on the runtime afterward. This separation of AI model delivery and container delivery has a few benefits on resource utilization. First, it will increase re-usability of an AI container. Instead of building a new container with different AI models and the same application logic for various edge resources, this method will enable a single AI container can be used on edges with different resources. Second, the network resources for the microservice will be optimized since only the container and specified model will be delivered to the edge.

In DECENTER, the inference will take place on either edge or the cloud, and the AI models can be used for inference at the edge side. Inference requires much less computing power than a training model in terms of processing. The computational complexity of the inference is just a set of the matrix multiplication operation. The advantages of inference at the edge side are as follows; The model does not require Internet access. And the service latency can be reduced by handling data directly to the model at the edge. Therefore, nowadays, the edge is trying to get the AI model trained in the cloud and infer directly. This chapter describes the new AI model repository in this project. The proposed server considers that the containerized application to be deployed on the edge does not include a static main AI model. Instead of this way, the edge device can import the necessary model directly from the AI model repository. The edge can import static AI model, but also can find and import the appropriate AI model based on the real-time needs by proposed. In addition, this server can handle partitioned AI model, which has been investigated in T4.1 and described in D4.1 [31].

7.1.1 AI model repository structure

In this chapter, the structure of AI model repository is presented. Figure 8 depicts how the application and AI model can be deployed to the edge devices. The figure on the left shows the way to deploy an application containing the AI model directly to the edge. The figure on the right shows the proposed method deploying the application without the main AI model on the edge. In this way, the edge can find the proper AI model from the AI model repository considering its computing resources.

The AI model repository acts as a pipeline to provide intelligence on individual edges. If edge does not know exactly which model to download, the repository server can search based on the requirements of the model and find the appropriate model.
As mentioned earlier, in DECENTER project, the trained AI model is used alone, but two or more split models are also used on the edge. N divided models are distributed to N edges respectively, and the output of one split model is used as the input of the other model. Therefore, above-mentioned characteristics should be reflected in model repository design. Related metadata should be also managed well so that lot of models can be easily reused and utilized. Therefore, DECENTER model repository server is designed to manage AI models and related information efficiently.

Figure 8. Two different ways to use the AI model on the edge side.

We applied a file system structure providing an interface to consistently find the proper AI model for the first version of the AI model repository. Developers can register their model and the system can manage a variety of models based on the pre-defined structure. As we mentioned, the prominent part of the advanced AI application is the flexibility caused by the model splitting. The AI application with partial AI model can be in conjunction with other edge’s corresponding application. Therefore, this repository is designed to properly save and deliverer the partitioned AI model to edge. Edge can access the repository server which has a structure by combining information of the model name, version, whether it is partitioned or not, and the number of the partition.

Figure 9 shows the stored structure of several variants of the modified VGG16 model. Each structure is divided into two parts: Full and Partial. Full is the directory where common whole models are stored. Partial is the directory to store the models which are split into more than two. Each model is managed by version. The subfolder contains the JSON file describing the
metadata for the stored model and the file to be downloaded from the edge. Based on this structure, the edge can access and download the proper model from the model repository server. The subdirectory named partial stores the data of the partitioned models. After a model is divided into a number of sub-models for collaboration between devices, this subdirectory is used for storage of split models. Partial directory subdivisions are grouped by version. The subdirectories contain the order of split models.

7.1.2 Semantics of the main elements

This repository provides the model specification file named "model_specification.json" associated to a specific AI model. It can be easily changed with any document-oriented database such as MongoDB. This specification file describes basic information about AI models such as model information, partial information, and train information. Model_specification.json file is created based on the pre-defined schema structure. In this stage, we registered and tested existing training models. We will then register customized models by individual use cases and make them available for use.

In the current stage, we registered and tested existing training models. We will then register customized models by individual use cases and make them available for use.

Model data type

The DECENTER AI model repository provides schema to describe the AI models. The serialization methods described above does not have explicit ways to include metadata for an AI model for the identification, and usually file names are used for identifying a model, which might be very inaccurate. In order to describe the AI model, each model should have metadata information. For example, the edge device is able to select an appropriate AI model based on the model cost, the model performance, the resources required by the model execution, and the ability of the privacy protection. The basic structure of metadata is defined as follows and can be extended in the future.

Figure 10. The basic structure of the AI model data type.
The AI model can be described using one model element having three main elements: model information, partial information, train information. The model information describes basic model information, and partial information describes the partition information of the model. Finally, train information describes the information used during model creation.

Figure 10 shows the structure of model data element, which is the root element of the schema.

- Model information has 9 sub-elements: general, history, specification, files, description, other information, usage, model input, and model output. It contains basic information for searching.
- Partial information has 2 elements: mode, partial num.
- Train information has only one element: DL libraries.

**Model information element**

Figure 11 shows the overall structure of model information type. Files element, description element and usage element can have multiple sub-elements which describe the detail information.

The files element describes all the files needed to run the model. The files element includes related file names. The server provides archive file which is composed of one or more model related files which are described in the files element. The description element describes additional information, such as basic information about the model provided or information to be used for the search. The usage element describes how this model can be used. An AI model can be used for various purposes such as classification, forecasting, and recommendation.

Complex elements other than the elements described above defines a semantic below.

Semantics of the generalType is:

<table>
<thead>
<tr>
<th>Name</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>This element describes the unique id to identify the AI model.</td>
</tr>
<tr>
<td>license</td>
<td>This element describes the rights associated with the model.</td>
</tr>
<tr>
<td>manufacturer</td>
<td>This element describes the manufacturer making the model</td>
</tr>
<tr>
<td>Name</td>
<td>This element describes the name of the AI model</td>
</tr>
</tbody>
</table>

Semantics of the history Type is:

<table>
<thead>
<tr>
<th>Name</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>date final available</td>
<td>This element describes the date the model was last registered.</td>
</tr>
<tr>
<td>historys</td>
<td>This element describes the relevant version information for the model.</td>
</tr>
<tr>
<td></td>
<td>If there are multiple versions, they contain information about all versions.</td>
</tr>
<tr>
<td>this version</td>
<td>This element represents the specific version number of this model.</td>
</tr>
</tbody>
</table>
Semantics of the specification Type is:

<table>
<thead>
<tr>
<th>Name</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>performance</td>
<td>This element describes the predictable performance when this model is used. It is not easy to test performance on all combinations of hardware during model operation. Therefore, it describes the minimum required hardware, and the performance when the model is running on this combination described.</td>
</tr>
<tr>
<td>size</td>
<td>This element represents the size of the model. When deploying and saving the model on the edge hardware, the import prerequisite is the size of free memory space which is bigger than the model size.</td>
</tr>
</tbody>
</table>

Semantics of the other Information Type is:

<table>
<thead>
<tr>
<th>Name</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>configuration parameters</td>
<td>This element describes the specific conditions for the input. At this stage, it describes specific conditions for the image. We need to extend the element scope.</td>
</tr>
<tr>
<td>Input type</td>
<td>This element represents the input type. It can be currently the image, audio, data, and sensors as an input type.</td>
</tr>
</tbody>
</table>

Semantics of the mode output Type is:

<table>
<thead>
<tr>
<th>Name</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>output specification</td>
<td>This element describes the specific information of output.</td>
</tr>
<tr>
<td>output type</td>
<td>This element describes the type of output.</td>
</tr>
</tbody>
</table>
Figure 11. Overall structure of model_information type.
7.2 Implementation environment

7.2.1 Implementation environment

A preliminary AI model repository has been implemented within the period. The repository is developed with the Django Web framework on the VM instance on the cloud. It is able to connect to this storage by general web interface and download the proper model. DECENTER microservices will be able to easily access models from the model repository using REST API for accessing models that are suitable for the microservices. Table 2 shows the detailed information of registered three models that are registered in the model repository. VGG16, YOLO v3, and customised AI model for indoor status prediction were used as candidate AI models.

Table 2. AI models used to deploy in DECENTER for the first year of the project.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VGG16</td>
<td>VGG16 is a convolutional neural network model proposed by K. Simonyan and A. Zisserman from the University of Oxford in the paper “Very Deep Convolutional Networks for Large-Scale Image Recognition”. This model achieves 92.7% top-5 test accuracy in ImageNet (A dataset of over 14 million images belonging to 1000 classes).</td>
</tr>
<tr>
<td>YOLO v3</td>
<td>You only look once (YOLO) is a state-of-the-art, real-time object detection system and one of the faster object detection algorithms. YOLO v3 uses a few tricks to improve training and increase performance, including multi-scale predictions, a better backbone classifier, and more.</td>
</tr>
<tr>
<td>Indoor PM10 prediction model</td>
<td>This model is a prediction model generated by KETI for UC 4 It can predict the future fine dust concentration based on the pattern of indoor air. It uses indoor and outdoor air data as input for a specific time period.</td>
</tr>
</tbody>
</table>

7.2.2 REST API Interfaces for AI model access

A microservice can download the model using the REST API. In DECENTER project, we propose a simple method to download the AI model using four parameters. The API is generated based on the model information: name, version, whether it is split, and the order of the model. This is an example of downloading the model. The base URL is the combination of the main host URL and “model”. The full URL address passes the 4 input parameters as query parameters as follows.
**BASE URL**

- base url is [http://[host_url]/model](http://[host_url]/model)
- Ex) [http://0.0.0.0:5000/model](http://0.0.0.0:5000/model)

**EXAMPLE ADDRESS**

- [http://[host_url]/model?model_name=VGG16&model_version=[v1.0]&partial_mode=Partial&partial_num=1](http://[host_url]/model?model_name=VGG16&model_version=[v1.0]&partial_mode=Partial&partial_num=1)
- [http://[host_url]/model?model_name=Yolo_v3&model_version=[v1.0]&partial_mode=Full&partial_num=0](http://[host_url]/model?model_name=Yolo_v3&model_version=[v1.0]&partial_mode=Full&partial_num=0)

**Path Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Required/Optional</th>
<th>Type</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model_name</td>
<td>Required</td>
<td>String</td>
<td>This server provides multiple AI models which can classify, predict the future status. First of all, micro-service needs to know the right model name to download it. Each microservice can download and manage the model based on the described information.</td>
<td>- model_name=Yolo_v3</td>
</tr>
<tr>
<td>Model_version</td>
<td>Optional</td>
<td>String</td>
<td>The AI model repository server manages the version of the AI models. The server must maintain a history of changes in the model because the AI model might be modified or improved. The microservice can download and use the appropriate version of the model.</td>
<td>- model_version=[1.0]</td>
</tr>
<tr>
<td>Partial_mode</td>
<td>Required</td>
<td>String</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>--------------</td>
<td>----------</td>
<td>---------------------------------</td>
<td>---------------------------------</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>In order to have one independent model on each microservice, the repository server should be able to provide a single AI model or a partitioned AI model. In this project, there are two types of model (Full/Partial) so that the microservice can cooperate with other devices.</td>
<td>partial_mode= Full</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Partial_number</th>
<th>Optional</th>
<th>Integer</th>
<th>-</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Partial Number is required to download the specific split model for the micro-service. In the case of Full mode model, its partial number is always 0. For split models, the order number can be set from 1 to the number of split models.</td>
<td>partial_number=1</td>
</tr>
</tbody>
</table>

7.2.3 Implementation results

As of now, we tested TensorFlow checkpoint files and pickle files as serialized model files. The checkpoint file has meta-information about the model, so it is easy to separate the model itself. We need to modify the original model to split it. Therefore, the checkpoint file is the best option to save and modify it. However, we will have to use the saved file format without metadata in the future because the checkpoint file having a lot of information is relatively large. We will test some other serialization formats. To validate the AI model repository, we registered several pre-trained models and tested the deployment process. Micro-service is able to connect to this storage and download the proper model remotely to derive an AI application. DECENTER application can use this API to download the model easily from the Model Repository. This repository server provides a simple API to access and download the right model for a specific microservice. This API address is designed to have four parameters to identify a model of interest: model_name, model_version, partial_mode, and partial_number.

We have created an AI model repository server that can provide real models. We have also developed a website that allows users to easily manage the registered model. Figure 12 shows the data distribution of input data type of the registered model. We registered five test models: three models for image classification and two models for analysing time-series data in Python Pandas’s DataFrame data type.
The repository server provides summary information about the registered model. We can see the name, version, updated date, purpose of use, model input and output, library used for training, and model partition information. Various information can be shown on the website (Figure 13).

7.1 AI model repository on model exchange management for Federated Learning

We tried to extend this system for federated learning. This chapter describes how to design and use this system for federated learning.

7.1.1 Storage in existing FL system

The role of storage in the existing FL system is as follows. The server is the main device of this system and stores the model and its associated base information into the storage. After that, it collects clients' information and selects appropriate clients. Each client needs the FL stored in persistent storage and sends it to the client. The client performs an individual FL task and sends the result to the server, which then collects the results directly and performs the
aggregation. The newly updated global model is stored in storage again and the above process is repeated.

We emphasize that, in the existing system, storage is used only for storing the FL server’s model as generated by it. Instead, the FL server’s application should perform most of the roles of a FL system, such as checking the status of clients, selecting the clients to be used, collecting important data, and delivering it to the clients, determining whether a sufficient amount of client data has been collected for aggregation or not.

Currently, the FL technology is limited to restricted use-case scenarios and is applicable to algorithms with limited infrastructure and devices. However, in the future, the number of various devices with sufficient computing power will increase, which would make the use of FL even more beneficial.

Another notion is, that, in the existing FL system, the server is the main actor and actively communicates with the client devices. The persistent storage, which is one of the assistive devices, is used only for storing stable and global information that the server sends.

The existing FL system might be reasonably useful when the number of the clients to be managed by the server is not too much. However, when this number increases significantly or when the number of candidate client devices and their respective specification is flexible, the management of all the resulting information can overwhelm the server.

It is inefficient for the server to manage the responses and the results from all the clients because it cannot estimate when the individual responses will arrive. In other words, all the responses from client devices which have their own environments and specifications are independent. Therefore, the server cannot predict the exact number and the arrival time of the final responses of clients. Besides, the aggregation task of FL can only start after all the responses from candidate clients have been collected.

Accordingly, it is necessary to reduce the dependency between the server and the clients so that FL can be practically used with various services. Besides, the system needs to be easily reconfigurable for various situations, such as dynamically changing the scope and number of clients used for FL.

7.1.2 The system architecture of AI model repository for FL

The figure on the right shows the comparison between the previous and the proposed system architecture. In the previous system, the FL clients receive necessary information directly from the server. The server uses the global AI model information that persists in the storage and delivers it directly to the FL clients. Therefore, the client does not need to communicate with the storage, as it stores only the global model information that is updated by the server.

However, in the existing method, the server and the clients depend heavily on each other, the meaning is that not all service tasks can perform before the next task can be finalized. The server must select a subset of clients among candidate devices that meet the eligibility criteria and objectives of the task. Additionally, it must wait until participating devices updates the reports. However, there may be cases in which the selected client does not send the adequate results or fails to send the result. It means that it can be inefficient because the server has to
manage all client connection even if the server initially selected the inappropriate client which produces the poor result.

Therefore, we propose a flexible FL system that enables each participating entity such as the server, the client, and the storage to depend less on each other and can collaborate easily. For this, the previous AI model repository was extended to support Federated Learning. The key improvement of the proposed AI model repository is to decouple the system in a way that operates asynchronously. In this way, this repository executes storage operation, so that there is less network dependency on one side, but also improved latency due to the ability of the storage system to operate faster.

7.1.3 The way to use the proposed repository for FL

The role of the AI Data management system in the FL system is to register and deliver the model by both server and client sides. We designed a decentralised AI data management system and implemented a web server to demonstrate its capabilities. The server was developed based on Python Flask and provides RESTful API to utilize the AI data management system. The basic requirements for uploading or downloading the model to the management system are Task_Name, Version, Model_Location, and Device_Name.

AI model repository can inform the performance and characteristics of each device, which helps to select FL participants. When the above four pieces of information are sent to the AI model repository, it transmits the corresponding AI data model information to the client or server, or stores them in the repository.

The way to use the proposed repository for FD is as follow:

- Define model information including model name, model version, and model location.
- Set the initial model version to 0.0.
- The version of each AI model can be described up to one decimal place.

<table>
<thead>
<tr>
<th>Name</th>
<th>Required/Optional</th>
<th>Type</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model_name</td>
<td>Required</td>
<td>String</td>
<td>Task_Name is a parameter to present the identity of each task to be solved using FL. Any server and client that has the ability to perform the specific task can be a candidate for participation. The data management system may inform the conditions of the device to perform each task. The servers and the clients can find their jobs by searching for information in the FL AI data management system. The model name to be registered must be unique. Check the name of the registered model.</td>
<td><code>model_name='FD_1'</code></td>
</tr>
</tbody>
</table>
**Model_version**  | Required | String  | Version is a floating-point number used by the server to update the global AI model data. When multiple servers and clients work together in one task, this version is the basis for managing the output.

Models with the same name can have multiple versions. The model version can be described up to the first decimal place.

| model_version | model_version | '0.0' |

**model_location**  | Required | String  | Model_Location informs about the device where AI data was first created. There are global models created on the server and local models created on the client. The server requests the local model, and the client requests the global model.

####server: The server registers an initial AI model and updates an advanced global model using the model created by clients. The updated model is registered again in the repository.

#### client: The client uses the initial model with local data to update the local model. It also uploads the updated parameters to the repository.

| model_location | model_location | 'server' |

**device_name**  | Required | Integer  | Device_Name represents a unique ID or name of each device.

The name of device which works for FL process.

| device_name | device_name | 'ID111' |

---

**Example) uploading server version model**

```python
model_info = {
    "model_name": "DECENTER_UC4_FL",
    "model_version": "0.0", # 0.0(initial)
    "model_location": "server", # server or client
    "device_name": "ID1102" # Unique server/client device name
}
# Upload URL: model/FL/DECENTER_UC4_FL/v0.0/server/ID1102_data.zip
```

# Create instance using FL_repository
FL = FL_model(repository_address)

# Set an model information attribute
FL.set_model_info(model_info)
```
D4.4: Final report and release on cross-border application data management and use-case specific AI solutions

# model upload code
# 1. Upload the model from server & client
# 1-1. Declare the address of the AI model to be uploaded.
# 1-2. Upload the model using API

```python
upload_file_name = "../test_model_storage/model_upload.zip"
FL.set_upload_file_name(upload_file_name)
FL.model_upload()
```

Example) uploading client instance

```python
# Create instance using FL_repository
FL = FL_model(repository_address)
# Set an model information attribute
FL.set_model_info(model_info)

# model upload code
# 2. Download the model for server & client
# 2-1. Declare the file name to be used when saving on local storage
# 2-2. Download the model using API

stored_file_name = "../test_model_storage/model_download.zip"
FL.set_stored_file_name(stored_file_name)
FL.model_download()
FL.model_save(FL_model)
```

8 Kubeflow and DECENTER Model Serving approaches

Within this section, Kubeflow will be compared to DECENTER in terms of features and functionalities available for users, especially as a model serving platform, and showing interoperability possibilities between both.

8.1 Kubeflow

According to its website, “The Kubeflow project is dedicated to making deployments of machine learning (ML) workflows on Kubernetes simple, portable and scalable”

https://www.kubeflow.org/docs/about/kubeflow/
8.1.1 Training workflow

In order to serve a model to users and stakeholders, this model needs to be previously trained with data in a proper environment. In this matter, throughout this section we will describe the process of using Kubeflow to train an AI model, which can be then served directly with Kubeflow services or embedded in one of DECENTER AI application services. In addition, this model can be created with the AI Package, as described in this document and previous deliverables like D4.3 and trained in the Kubeflow platform. As you may see, there are interoperability possibilities between both.

Kubeflow works at the pipeline level. According to the documentation\(^\text{13}\), a Kubeflow pipeline is a description of the steps and interactions of a ML workflow, plus the parameters that are required to run it. The pipeline components are just a step in the workflow, each one performs a specific task and has inputs and outputs that can be redirected from/to other components. The task can be performed by a Docker container or a Python process. They need to follow the Kubeflow’s SDK guidelines\(^\text{14}\). In addition, since Kubeflow is running on top of Kubernetes, the pipelines can also manipulate Kubernetes resources\(^\text{15}\) as part of its execution, as for example creating Persistent Volumes to store some data.

To create a pipeline, users of Kubeflow need to follow these steps:

1. Install Kubeflow pipelines sdk.
2. Create a docker container or a python script for each component\(^\text{16}\). This container or script represents your task and can have some input parameters and outputs.
   a. When creating the docker image, use `ENTRYPOINT` instead of `CMD`, as any arguments you pass to the docker run command will be passed to the entry point of the image.
   b. Each component output should be a string and should be written in a separated text file. This text file shall be shared in a Persistent Volume Claim with other components.
3. Create a Python function representing your component: describing the inputs (arguments that will be passed to the docker container or the python script), outputs and the functionality (by indicating your docker image or your python script).
4. Create a python function representing your pipeline following the Kubeflow pipelines SDK, where you can indicate the inputs of the pipeline.

Since the AI Package provides functionalities to create an AI model and encapsulate it into an AI method, it can be directly train on Kubeflow, just by encapsulating the training of the AI model into a Kubeflow pipeline, as it has been explained. The pipeline was created first with a shared Volume so that the data and the model can be shared between components (steps) of the pipeline, as shown in Figure 14.

\(^{13}\) https://www.kubeflow.org/docs/pipelines/sdk/build-component

\(^{14}\) https://www.kubeflow.org/docs/pipelines/sdk/install-sdk/

\(^{15}\) https://www.kubeflow.org/docs/pipelines/sdk/manipulate-resources/

\(^{16}\) https://www.kubeflow.org/docs/pipelines/sdk/sdk-overview/
Then, the pipeline includes another component with the actual docker container that uses the AI package to create and train a model, as shown in Figure 15. When this step finishes, the model is stored in the shared volume, and is uploaded to the repository in the last step.

After the pipeline is created, the next step is just executing it. In order to do so, users of Kubeflow must create an experiment, which represents the run of the pipeline, as shown in Figure 16.
KubeFlow pipelines have been tested in another use case (UC4) of DECENTER. In this test, both parallel and sequential pipelines are constructed to see the effect of parallelization. The UC4 of DECENTER, is related to the Ambience Intelligence and comprises three AI-based microservices, namely:

- **Face detector (FD):** is responsible to extract the portions of frames that contain faces of the person standing in front of a camera;
- **Feature extractor (FE):** takes batches of 160x160 face images as input and extract their feature vectors;
- **Member verifier (MV):** takes the feature vectors and returns the membership of the person.

In our analysis, we only consider the FE and MV components, where the latter is divided into Train-MV and Test-MV: the first is used to train the neural network, while the second takes as input the trained network and is used at inference time for membership verification (Error! Reference source not found.).

We compared two different Kubeflow pipelines (or workflows) for our use case, namely **Parallel and Sequential.**
Parallel: pipeline that exploits parallel execution of tasks: four FE instances are executed at first, followed by two Train-MV components and by one Test-MV. The parallelization is done as shown in Figure 18: the dataset is divided in four parts (matching the four groups of people in the dataset, two member groups A and B, one non-members and one test group), leading to four FE components. Then training is divided in two components because the training phase generates two models: one made to recognize the members of group A and one members of group B.

 Sequential: a pipeline with no parallelization. There are only three components that execute their tasks in a sequential way (Figure 19).

8.1.1.1 Comparison of the execution time

Parallel execution of the pipelines is a major benefit of Kubeflow. Here we measure it in terms of execution time by comparing the two approaches: parallel and sequential.

Table 3: Execution time (mm:ss)

<table>
<thead>
<tr>
<th></th>
<th>Parallel</th>
<th>Sequential</th>
<th>Bare container</th>
</tr>
</thead>
<tbody>
<tr>
<td>FE</td>
<td>2 : 14</td>
<td>3 : 44</td>
<td>0 : 54</td>
</tr>
<tr>
<td>Train-MV</td>
<td>2 : 07</td>
<td>2 : 46</td>
<td>0 : 53</td>
</tr>
<tr>
<td>Test-MV</td>
<td>0 : 03</td>
<td>0 : 02</td>
<td>0 : 02</td>
</tr>
</tbody>
</table>
From Table 3 we can see how execution times differ between the different pipelines. As foreseen, parallelization has a beneficial effect on execution time. However, we see that the gain in processing speed is not as high as expected. Specifically, given the four instances of FE and the two instances of Train-MV in the parallel execution, a reduction of approximately four times and two times was expected for the FE and Train-MV components respectively. This effect is due to the overhead introduced by the Kubeflow mechanism that manages the pipelines. This overhead can be estimated by comparing the time of the sequential approach with the last column in Table 3, which reports the execution times of the modules executed in sequential fashion without Kubeflow. Such an overhead is the price to pay for the automation of AI tasks brought by Kubeflow. Nevertheless, as we can understand from the results in Table 3, this overhead can be partially mitigated by the parallel execution of tasks.

8.1.2 Kubeflow as a Model Serving platform

With a trained model, now the next step would be to offer this model to UC users and other stakeholders. As explained in D5.1, Kubeflow offers possibilities to do this with its Kubeflow Serving service as well as the Seldon Core serving service. KFServing and Seldon Core Serving are considered multi-framework model serving systems since they can serve models of different types of Machine Learning frameworks like Tensorflow or Pytorch.

Figure 20: Technology stack of KFServing, the supported multi-framework model serving system of Kubeflow (extracted from KFServing GitHub Repository17)

---

17 https://github.com/kubeflow/kfserving
To serve a model, users of Kubeflow must create a microservice that serves the model using the KFServing Python SDK\(^\text{18}\). The serving of the model is subject to be configured with the different features that Kubeflow offers. The features that can be configured are collected in Table 4. As explained in the next section, the models for DECENTER are served directly with the DECENTER model serving platform, as it is designed to operate within the DECENTER Fog platform and monitored directly with the AI Package and Repository functionalities.

### 8.2 DECENTER model serving

As described in section 4 of D4.3, DECENTER provides the AI Package for AI methods containerization into microservices, in other words, DECENTER model serving is done based on the AI Package functionalities.

Once an AI model, encapsulated into an AI method, has been designed with the AI Package and it has been trained, it is containerized as a deployable microservice. The process of containerization in DECENTER parts from a base Docker image provided within the AI Package, which encapsulates the libraries needed to properly run the model, as shown in Figure 21.

![Software stack for AI application and virtualization](image)

Figure 21: Software stack for AI application and virtualization

More information about this containerization and creation of microservices is described in D4.3, along with the features that the AI Package offers to serve models.

In this regard, after evaluating the characteristics of Kubeflow's model serving approaches, we have come up with a set of features that may be relevant to DECENTER’s model serving approach, and which can be added to its already existing characteristics. This set of features can make DECENTER model serving more complete, especially in the context of decentralized scenarios and use cases.

In Table 4 the supported features per serving platform can be observed. Those features are highlighted in green. On the other hand, the features that are relevant for DECENTER as a model serving platform are highlighted in yellow.

Table 4: Kubeflow Serving, Sheldon Core and DECENTER Serving Supported features

<table>
<thead>
<tr>
<th>Feature</th>
<th>Sub-feature</th>
<th>KFServing</th>
<th>Seldon Core</th>
<th>DECENTER</th>
</tr>
</thead>
<tbody>
<tr>
<td>Framework</td>
<td>TensorFlow</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>XGBoost</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td></td>
<td>scikit-learn</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td></td>
<td>TensorRT</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ONNX</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>PyTorch</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Inter-pod Graph (optimization)</td>
<td>Transformers</td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Combiners</td>
<td>Roadmap</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Routers including test A/B and MAB(^{19})</td>
<td>Roadmap</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Splitters</td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Intra-pod Graph (optimization)</td>
<td>Plugable model server pipeline</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Analytics</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Monitoring</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Scaling</td>
<td>Knative (serverless)</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td></td>
<td>GPU AutoScaling</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>HPA</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Custom</td>
<td>Container</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Language Wrappers</td>
<td></td>
<td>✓</td>
<td></td>
</tr>
</tbody>
</table>

\(^{19}\) MAB (Multi-armed bandit) [https://en.wikipedia.org/wiki/Multi-armed_bandit](https://en.wikipedia.org/wiki/Multi-armed_bandit)

\(^{20}\) Pluggable, and complete pipeline for model serving through a framework for pre-processing, prediction, post-processing and explainability out of the box.
<table>
<thead>
<tr>
<th>Multi-Container</th>
<th>✓</th>
<th>✓</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rollout</td>
<td>Canary</td>
<td>✓</td>
</tr>
<tr>
<td>Shadow</td>
<td>✓</td>
<td>X</td>
</tr>
<tr>
<td>Service mesh</td>
<td>Istio Connect</td>
<td>✓</td>
</tr>
<tr>
<td>Istio Secure</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Istio Control</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Istio Observe</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Management</td>
<td>Versioning</td>
<td>✓</td>
</tr>
<tr>
<td>Updater</td>
<td>✓</td>
<td></td>
</tr>
</tbody>
</table>

The features that can be relevant for DECENTER are explained in this section:

- **Analytics**: This feature conforms the analysis and understanding of the model. Not only in terms of exactitude on its outputs, but also on why the model is giving such output according to the input it was given. There can be extracted two sub-features:
  - **Explanations**: This allows for model’s interpretation and inspection to explain its behavior against the inputs that it is receiving.
  - **Monitoring**: This sub-feature looks out for the accuracy of the model against the incoming requests, providing insights of to what inputs its accuracy is better or worse.

- **Scaling**: This feature watches over the performance, for instance in terms of CPU consumption or memory usage, of the container that is exposing the model against incoming requests.
  - **Horizontal Pod Autoscaling**: This feature contemplates the scaling up of the deployment of the model in case of an increasing number of requests, by adding more similar containers in parallel, serving the same model. Kubernetes (or the deployment platform) should be in charge of distributing requests equally among deployments. If the requests decrease, it also contemplates the removal of redundant containers.

- **Rollout**: This feature watches over the deployment of a new version of the model. In case it does not behave as expecting, it transparently redeploys the previous version of it and removes the deployment with the new version. There are two strategies:
  - **Shadow**: Deploying the new version of the model for testing, in parallel with the previous version, to identify if it meets the requirements. In case it does not, the solution is just to remove the new deployment.
- **Canary**: Deploys the new version of the model and watches over its performance. In case it does not behave as expected, the deployment is removed and the previous version, which was kept in memory, is deployed again.

In any case, as described in D4.3, DECENTER provides some unique features in comparison with Kubeflow Serving and Sheldon Core Serving:

- **Splitters**: ability to split a deep neural networks (DNN) into several containerized sets of layers. It means dividing the ML pipeline captured by a single DNN to be deployed in different microservices, which will be invoked sequentially at runtime.

- **Model versioning**: ability to understand and work with the concept of version of models.

- **Model updater**: ability of the model server to decide when it is the best moment and/or required to update the ML model, due to reasons such as obsolescence or underperformance.

As described in Chapter 7 of this document, the AI Model Repository had defined data types to describe and identify AI model to make it easy for AI developers to manage the developed AI models. The AI model Repository provides RESTful APIs to access those functionalities, and moreover it can be easily turned to a Custom Resource Definitions (CRD) in Kubernetes. When it’s turned to a CRD, the AI application developers can monitor a model of interest in more cloud-native way and can easily update the model which is used in AI microservice with a simple implementation of custom controller.

As a conclusion, we can say that, even though Kubeflow provides a remarkable platform for model training and serving, especially in the form of supporting lots of frameworks and tools, DECENTER serving is skillful enough to support the deployment of AI services. The AI Package has been designed to cover all the end-to-end process of an Artificial Intelligence operation and is built to support the deployment of AI applications on decentralized scenarios by providing insights on designing AI microservices that deploy and serve AI models. Not only that, but also it was designed to run on the developed DECENTER Fog Platform that improves the Kubernetes platform.

**9 DECENTER AI solutions and implementation guidelines**

In DECENTER project, we have investigated how to build a cloud-native AI application on distributed infrastructure. Though we were able to provide number of tools along with the platform, it is true that there are many considerations to design and deploy an end-to-end AI service on distributed infrastructure since it is related quite various technologies from Cloud to AI. DECENTER project has acquired a good insight on them while realising four of our use cases with use-case specific AI solution design, which has been summarized in Table 5.

<table>
<thead>
<tr>
<th>Building Blocks</th>
<th>Description</th>
<th>Application in UCs</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AI Optimization methods</strong></td>
<td>AI model optimized to use less computing resource, to make it suitable for edge devices. DECENTER has investigated the following methods</td>
<td>The AI optimization methods are useful in use cases where small resources such as embedded devices are used.</td>
</tr>
</tbody>
</table>


• Model Split method
• Intermediate Data Compression
• Model Pruning
• Quantisation and binarisation

**UC1:** The methods can be used for building lightweight AI models to be deployed on the edge devices in roadside.

**UC2:** The methods can be used for building lightweight AI models to be deployed on Robots.

**UC4:** The methods can be used for edge devices which is connected to a camera.

**Digital Twin**
The data which have been fed or generated by an AI microservice are stored and analysed to Digital Twin representation and semantic representation of data. The relations between data are analysed and provided to the user as a knowledge base. User can access those data to present Digital Twin application.

**UC1:** The road condition can be presented with Digital Twin. More details on semantic relationship between the entities can be provided as well.

**UC2:** The status of warehouse and robots can be presented with Digital Twin, along with semantic representation of entities in the warehouse.

**UC3:** The status of construction site and semantic relations between the entities can be presented with Digital Twin.

**UC4:** The occupation of meeting rooms and relevant information can be provided with Digital Twin.

**Data Management (AI Model Repository)**
AI Model Repository provides methods for AI model management for AI service. Also it works with BlockChain and Smart Oracle to guarantee to manage access to those data.

**UC3:** The AI model to identify worker in a construction site can be deployed only to the authorized resources with this method.

**UC4:** The AI models to identify membership of a person can be deployed only to the authorized resources.

### 9.1 Guideline to use DECENTER facilities

This guideline describes how to design and deploy your AI service from an existing AI model(s) in eight steps. First the resources for existing AI model(s) are measured, and then applied to the microservice architecture. Those AI models can be implemented as microservices with DECENTER AI Package, and with a few modifications. With that you can design interaction with additional DECENTER facilities such as Digital Twin or Resource Monitoring, along with interfaces for other microservices. Those interfaces will be described in Kubernetes-compatible configuration files and will be used for the deployment.

- **Prerequisite:** AI Model(s): You need AI model(s) to implement AI application.

First you need AI model(s) for your AI application implementation. They need to be stored in serialized format. If you intend to do some kinds of training on the resources, they can be used as trainable files such as checkpoint file in Tensorflow. If there is no need of training and you want to use inference on the resources only, then it would be better to freeze the AI model and save it to serialized formats such as ONNX or NNEF.

*After this step, you will have metadata of the model - Model Name and Model Version.*

- **Step 1:** Measure AI resources

To design your AI application, you need to know what kinds of AI-specific resources that you need. This is one of the most important decision to make while designing your AI application with considerations on the performance. Basically, an AI model is a complex set of various...
computation methods such as convolution and/or ReLU, and the resources required are decided with respect to the performance requirements.

What kind of resources and how much of them are needed to run an AI model is a very common question for AI application design. The truth is it cannot be answered without specific performance requirements. Suppose that you want to detect an object from HD video stream (1080p) with Yolo v3 model. You can apply some kind of GPU resources for this one, for example Nvidia 1080ti. With this resource your AI application can process HD-quality object detection in real-time (30 frame per second). Or if your application does not require that fast computation, then you can go with the CPU device only. With Intel i7 CPU, it can proceed object detection around 10fps. So, the first thing to be decided is a performance requirement for the AI application.

When the performance requirements are decided we can choose proper AI-specific resources. Some of useful criteria for choosing resources are listed here.

- **Computational Resources**: Many AI-accelerator devices have built-in support for parallel data processing, and usually performance of this parallel processing is referred to the number of units (tensor in Nvidia module). If you have a complex and big-sized model, it needs acceleration device with more tensor units.

- **Memory for acceleration device**: AI accelerator devices needs to load the AI model along with the data to the memory which are dedicated to them. Usually, acceleration devices have dedicated memory for the computation and data processing.

- **Memory for CPU**: Even though there is an acceleration device, some processing requires CPU operations. Grabbing a frame, resize and pre-processing of it are usually jobs of CPU. If you want to process the data in real-time, it needs sufficient memory for CPU operations as well.

- **Power consumption**: Acceleration devices consumes more power. Recent acceleration devices from Nvidia consumes more than 200W, and it goes more than 300W if it is a GPU PCI-compatible card. If you want to run your AI service on a power-constrained environment, then an embedded accelerator can be an option. Jetson series from Nvidia or TPU module Coral consumes less than 20W for AI model computation.

There are a few tools to help measuring how much resources that your AI application is using provided by acceleration device vendors. For example, you can check how much GPU resources are being used with `nvidia-smi`, if you’re using GPU from Nvidia, and with `tegrastats`, if you’re with Jetson series.

After this step, you will have a list of acceleration resource lists: type of acceleration device, name of acceleration device, memory of acceleration device – the same properties defined in DECENTER’s deliverable D3.3 [31].

- **Step 2**: Design your AI as microservices

Since the AI models and the resources for them are ready, now you can design your AI service with microservice architecture. The microservice architecture refers a computing architecture which arranges an application as collection of loosely coupled services, and with DECENTER the AI application can be built as one of those loosely coupled services. DECENTER
investigated how to decompose an AI application into microservices (D4.1) and also provides good examples of designing AI service with microservice architecture (D4.3).

With DECENTER, the AI will be transformed to an AI-microservice. That AI-microservice will be dedicated to the computation of AI. It can be connected to other microservices such as web front-end, database, IoT platform, etc. DECENTER platform supports edge computing, so you can specify which region you want to deploy a specific service (or DECENTER can decide it for you).

After this step, you will have a structure of your AI application in microservice architecture, with cloud-edge requirements for each microservice.

- **Step 3:** Optimize your AI models (optional)

You can use the AI model as-is, but it would improve usability of your AI model or microservice if you can reduce the resource requirement of it. For example, if your AI microservice requires just a few megabytes of memory with embedded GPU, it means that it can be deployed onto any resources from embedded devices to the cloud resources. Or if your AI microservice requires V100 GPU and hundred gigabytes of GPU memory, then the candidate resources for the deployment will be highly restricted.

In DECENTER, we have investigated and implemented a few AI model optimization methods.

- **AI Model Partitioning:** If the size of an AI model is too big, you can split that AI model into multiple parts, and connect them together in a microservice architecture with DECENTER. Also DECENTER provides a good compression method to decrease the size of intermediate data transferred between partitioned AI models. Details are described in D4.1 [32].

- **Pruning:** With this Pruning method you can eliminate node of less importance, to reduce computational load of AI service. DECENTER has shown that this pruning method can be applied to one of our use cases, effectively reducing resource requirements while preserving the accuracy in a moderate level.

- **Quantization and Binarization:** The size of an AI model can be reduced further if the parameters of a model can be quantized. DECENTER is working on realization of Quantization and Binarization in Y3, and will deliver insights width DECENTER.

After this step, you’ll have optimized set of AI models.

- **Step 4:** Build AI Microservices

Now the architecture and models are ready, and we can move on to building an AI Microservice. DECENTER provides an AI package, which helps making AI microservice from AI application. The benefits provided by the DECENTER AI Package is as follows:

- **AI model management.** It works with AI model repository to find a suitable model.

- **Network Interfaces.** AI Microservice needs network interface APIs, since it needs to be coupled with other microservices. DECENTER AI Package provides network interface APIs to control and manage AI in it, and it is extendible.

This step consists of a few sub-steps.
1) Finding base container.

Since this AI microservice is containerized, it needs to find a suitable base container for it. In our use case implementations, many AI microservices are using GPUs from Nvidia, and corresponding containers can be found on Nvidia cloud. Here is some helpful repository list to find suitable base container.

- Official Tensorflow Docker Hub contains Docker images which contains TensorFlow\(^{21}\).

- Nvidia GPU Cloud (NGC): Nvidia provides docker container images\(^{22}\) to be used with their GPU devices. Containers for Jetson series can be found there as well.

- Containers for ARM: Docker Hub provides Docker images for ARM64 devices\(^{23}\). If you’re considering some ARM devices such as Raspberry Pi, this is a place to take a look.

2) Build container image with DECENTER Package

DECENTER provides reference Docker files to build a container image. You can install this package just like the other dependencies. Here is an example of installing DECENTER package and OpenCV in a container.

```
FROM nvidia/cuda:10.2-cudnn7-devel-ubuntu18.04
RUN apt-get update
RUN pip3 install scikit-build cython setuptools
# Install DECENTER AI Package
ADD decenter-0.6-py3-none-any.whl /
RUN pip3 install /decenter-0.6-py3-none-any.whl

# Install OpenCV
RUN pip3 install opencv-python-headless opencv-contrib-python-headless
```

3) Refactor MyModel.py to reflect your AI application. There are three functions for refactoring in skeleton of MyModel.py.

---

\(^{21}\) https://hub.docker.com/r/tensorflow/tensorflow/

\(^{22}\) https://ngc.nvidia.com/catalog/containers?orderBy=scoreDESC&pageNumber=0&query=jetson&quickFilter=containers

\(^{23}\) https://hub.docker.com/u/aarch64/
D4.4: Final report and release on cross-border application data management and use-case specific AI solutions

```python
class MyModel:
    def __init__(self):
        pass
    def load_ai_model(self, filename):
        pass
    def compute_ai(self):
        pass
```

- **Function `__init__`**: place initialization methods in this function.
- **Function `load_ai_model`**: place code to load your AI model onto memory, with respect to your serialization method. For example, you can define network structure and load parameters in this function if it's TF checkpoint.
- **Function `compute_ai`**: This is the main function which process AI computation. Place your AI logic in this function. The input URL will be delivered to this function as an argument, and this function returns result of AI computation.

With this simple refactoring, you can use your AI model as a microservice. DECENTER AI package has built-in network interfaces to call the above functions to control and manage AI lifecycle. In addition, you can configure your AI service interaction with other microservices as follows.

- **Step 4-1: Design additional interfaces**

DECENTER AI package provides common set of interfaces to control and manage AI, which is described in D4.1. Those interfaces can be used set or get input and output data of AI, and control AI operation. If you want to add additional control of your AI microservice, you can add some more interfaces on main.py.

```python
flaskapp = msg_handler.get_flask_app()
@flaskapp.route('/set_fps')
def set_fps():
    fps_val = request.args.get('fps')
    logging.info('set fps received with value: ' + fps_val)
    my_model.set_fps(float(fps_val))
    return 'fps set'
```

The code above shows example of adding FPS variable and corresponding interfaces to my_model class. DECENTER AI package uses flask app, so first it needs to retrieve flask handler from DECENTER message handler, and then you can simply add interfaces as shown on the above example.

- **Step 4-2: Design AI-specific metrics for resource monitoring**

DECENTER provides Resource Monitoring, which can be used in SLA implementation. There are two ways to feed AI-specific metrics to the Resource Monitoring module from an AI microservice. One is to follow Prometheus implementation, injecting sidecar container for
resource monitoring with Istio. The other one is to configure AI microservice to deliver AI-specific metrics to a specific Prometheus server. You can deliver the AI-specific metrics with MyModel.py implementations. Please refer corresponding deliverable for details (D3.4)

- **Step 4-3: Design Digital Twin interaction**

DECENTER provides Digital Twin to make use of data and feature generated in AI processing. If you want to make use of data, feature and/or analysis result with Digital Twin, you can deliver the result to it by configuring output destination of AI microservice in DECENTER AI Package. DECENTER AI Package can have multiple output (analysis result) destinations, and simply adding DT destination will resolve this one. For details on Digital Twin implementation with SensiNact, please refer deliverable D4.5, which is scheduled on M36.

*After this step, you will have AI microservice containers.*

- **Step 5: Register AI Containers and AI Models**

In DECENTER, AI container and AI model are managed independently. This independent management will help optimize network usage and management. If the model is updated, only updated model will be transmitted from AI model repository to the running container instead of re-deploying the whole container. Since the size of model is small when it’s compared to that of a container, the network bandwidth usage will be reduced, further this will help reduce microservice interruption since this happens on a running container and no re-start or re-deployment are required.

In this step, you register the AI microservice container from Step 5, and register the AI models from either Step 3 or Step 1, depending on whether optimization method has been applied. For AI model repository, please refer to D4.2 or D4.4.

*After this step, you will have URL of your microservice in container repository, along with description of AI models in AI model repository.*

- **Step 6: Describe AI-specific Resources for AI Microservice deployment**

The software for the deployment – microservice and AI model – are prepared until Step 5. From Step 6, the procedure to describe Kubernetes deployment are described. Actually, these steps are not required if you’re going to deploy your AI service using AppComposer and FogAtlas of DECENTER, however describing the YAML description may help developers to understand how it works.

In AI microservice deployment, you need two files. The first one is a YAML file describing deployment object of Kubernetes, and the other one is another YAML file describing configuration of the microservice. In DECENTER we use Kubernetes ConfigMap to describe the configuration.

- Deployment YAML file will contain the container URL of container repository, as described in Step 5.

- Configuration YAML file will contain the model metadata as described in Step 5, along with input and output destination as designed in Step 2.
In the following is an example of deployment YAML file. The location of container is described in "image" keyword, and the ConfigMap object which describes the configuration for this microservice is written in "configMapRef". This example specifies a resource which has a specific AI-related resources with "nodeSelector". This is not needed if you use AppComposer and deployment algorithm in DECENTER.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: fd-deployment
  namespace: uc4
  labels:
    app: face-detector
spec:
  replicas: 1
  selector:
    matchLabels:
      app: face-detector
  template:
    metadata:
      labels:
        app: face-detector
    spec:
      containers:
        - name: face-detector
          image: keticmr.mynetgear.com:22500/uc4-fd:v1.2
          imagePullPolicy: IfNotPresent
          env:
            - name: MY_APP_CONFIG
              valueFrom:
                configMapKeyRef:
                  name: uc4-fd-config
                  key: appconfig
          nodeSelector:
            kubernetes.io/arch: amd64
            accelerator: nvidia.2080ti
            region-name: keti

The following is an example of configuration YAML. The configuration is written in appconfig variable with JSON format. It has three types: input, output and ai_model. Here the design of the microservice architecture is reflected in input and output, and the metadata of AI model in Step 5 is written in ai_model.
You can use these two files to deploy the AI-microservice for any Kubernetes-compatible platform, including DECENTER. This step is not required if you deploy your service with AppComposer and/or DECENTER platform.
10 Conclusion

This deliverable delivers the final results on task T4.3, which aimed on delivering data management mechanisms for AI applications in cross-border scenarios; and report on the activities on task T4.4 for the period M24-M30. Hence, this deliverable provides the final scenario and system architecture for cross-border data management alongside its implementation details and the compliance with GDPR and PIPA regulations. In the context of T4.4 it delivers updates on the AI model repository and detailed information on the implementation environment and results.

Our activities within T4.3 and T4.4 from WP4 managed to achieve the following:

1. The list of trust attributes in what it concerns cross-border data management are extended and analysed in the context of the DECENTER's Fog Computing Platform. The goal of this achievement was to analyse key aspects and attributes to trust that are important for smart applications and environments. In comparison to the previous deliverable, here the attributes were organized in two subsets: fundamental and derived trust attributes.

2. Key data protection regulations in the EU and Korea are considered in the context of cross-border data management. Hence, the GDPR and PIPA regulations were studied to determine key roles in the regulations and the articles that regulate the rights for each role.

3. Detailed cross-border AI model management scenario are improved and updated for final implementation and integration. Apart of improving the final design, the cross-border management scenario has been complementary updated to comply with the GDPR and PIPA regulations.

4. Specific use case for AI models cross-border management was developed, implemented, and tested. The use case was used to evaluate and test all the software components that were developed in the scope of the research and development activities for the cross-border data management scenario.

5. Several blockchain-based mechanisms were designed and developed to serve as basis for the implementation for cross-border data management. Hence, multi-party Smart Contracts and Smart Oracles were prepared to provide a safe, secure, and trustworthy operations over AI-models in the cross-border scenario.

6. A newly developed repository for AI models was integrated in the cross-border data management scenario. The developed AI model repository acts as a pipeline to provide intelligence on individual edges that allows searching and fetching AI-models based on given requirements.
References


**Abbreviations**

WP Work Package
SLA Service Level Agreement
SLO Service Level Objective
SMI Service Measurement Index
ISO International Organization for Standardization
IT Information Technology
QoE Quality of Experience
QoS Quality of Service
K8S Kubernetes
ML Machine Learning
AI Artificial Intelligence
SaaS Software-as-a-service
PaaS Platform-as-a-service
IaaS Infrastructure-as-a-service
CPU Central Processing Unit
GPU Graphics Processing Unit
RAM Random Access Memory
IoT Internet of Things
AWS Amazon Web Services
VM Virtual Machine
GUI Graphical User Interface
CLI Command Line Interface
API Application Programming Interface
REST Representational State Transfer
REB Resource Exchange Broker
<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB</td>
<td>Database</td>
</tr>
<tr>
<td>VPN</td>
<td>Virtual Private Network</td>
</tr>
<tr>
<td>UML</td>
<td>Unified Modelling Language</td>
</tr>
<tr>
<td>DAG</td>
<td>Directed Acyclic Graph</td>
</tr>
<tr>
<td>NGI</td>
<td>Next Generation Internet</td>
</tr>
<tr>
<td>BC</td>
<td>Blockchain</td>
</tr>
<tr>
<td>DLT</td>
<td>Distributed Ledger Technology</td>
</tr>
<tr>
<td>MTBF</td>
<td>Mean Time Between Failures</td>
</tr>
<tr>
<td>MTTR</td>
<td>Mean Time To Repair</td>
</tr>
<tr>
<td>MQTT</td>
<td>Message Queuing Telemetry Transport</td>
</tr>
<tr>
<td>HTTP</td>
<td>HyperText Transfer Protocol</td>
</tr>
<tr>
<td>IDE</td>
<td>Integrated Development Environment</td>
</tr>
<tr>
<td>OSGI</td>
<td>Open Service Gateway Initiative</td>
</tr>
<tr>
<td>ETH</td>
<td>Ethereum</td>
</tr>
<tr>
<td>PoW</td>
<td>Proof of Work</td>
</tr>
<tr>
<td>BC</td>
<td>Blockchain</td>
</tr>
</tbody>
</table>