TEADAL

D4.2 STRETCHED DATA LAKES

SECOND RELEASE REPORT

Revision: v.1.0

Work package	WP 4
Task	Task 4.1, 4.2, 4.3
Due date	31/10/2024 (extended)
Submission date	31/10/2024
Deliverable lead	IBM
Version	1.0
Authors	Ronen Kat (IBM), Ofer Biran (IBM), Katherine Barabash (IBM), Mohamed Mahameed (IBM), Temesgen Magule Olango (ALMAVIVA), Hine Samantha (ALMAVIVA), Sergio Sestili (ALMAVIVA), Andrea Falconi (Martel), Sebastian Werner (TUB), Sepideh Masoudi (TUB), Fernando Castillo (TUB), Eduardo Brito (CYB), Gabriele Cerfoglio (Martel), Rizkallah Touma (i2CAT), Bruno Feitas (UBIWHERE)
Reviewers	Bruno Feitas (UBIWHERE) Eduardo Brito (CYB)
Abstract	Technical summary of the demonstration of the control plane, data management, and trustworthy data flows working together in selected scenario, addressing 50% of the relevant KPIs
Keywords	Control plane, Data lake, multi-cloud, multi-cluster, data pipelines

Grant Agreement No.: 101070186
Call: HORIZON-CL4-2021-DATA-01
Topic: HORIZON-CL4-2021-DATA-01-01
Type of action: HORIZON-RIA

Document Revision History

Version	Date	Description of change	List of contributor(s)
V0.1	26/06/2024	Outline	Ronen Kat (IBM)
V0.2	19/08/2024	Integrated the first partner contributions round	Ofer Biran (IBM), Temesgen Magule Olango (ALMAVIVA), Andrea Falconi (Martel), Sebastian Werner (TUB), Sepideh Masoudi (TUB), Hine Samantha (ALMAVIVA), Eduardo Brito (CYB)
V0.3	08/10/2024	Ready for the second contributions round	Katherine Barabash (IBM)
V0.4	16/10/2024	Integrated the second contributions round	Katherine Barabash (IBM), Ronen Kat (IBM), Mohamed Mahameed (IBM), Hine Samantha (ALMAVIVA), Temesgen Magule Olango (ALMAVIVA)
V0.5	22/10/2024	Integrated the third contributions round	Katherine Barabash (IBM), Ronen Kat (IBM), Mohamed Mahameed (IBM), Temesgen Magule Olango (ALMAVIVA), Gabriele Cerfoglio (Martel), Eduardo Brito (CYB), Sergio Sestili (ALMAVIVA), Rizkallah Touma (i2CAT)
V1.0	28/10/2024	Ready for internal review	Katherine Barabash (IBM), Ofer Biran (IBM), Temesgen Magule Olango (ALMAVIVA), Sergio Sestili (ALMAVIVA), Hine Samantha (ALMAVIVA), Rizkallah Touma (i2CAT), Bruno Feitas (UBIWHERE), Fernando Castillo (TUB)
Final	31/10/2024	Ready for submission	All authors and reviewers

DISCLAIMER

Funded by the European Union

Project funded by
Schweizerische Eidgenossenschaft
Confédération suisse
Confederazione Svizzera
Confederaziun svizra
Swiss Confederation
Federal Department of Economic Affairs, Education and Research EAER
State Secretariat for Education, Research and Innovation SERI

Funded by the European Union (TEADAL, 101070186). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union. Neither the European Union nor the granting authority can be held responsible for them. This work has received funding from the Swiss State Secretariat for Education, Research and Innovation (SERI).

COPYRIGHT NOTICE

Project funded by the European Commission in the Horizon Europe Programme		
Nature of the deliverable:	**R**	
Dissemination Level		
PU	*Public, fully open, e.g. web (Deliverables flagged as public will be automatically published in CORDIS project's page)*	✓
SEN	*Sensitive, limited under the conditions of the Grant Agreement*	
Classified R-UE/ EU-R	*EU RESTRICTED under the Commission Decision No2015/ 444*	
Classified C-UE/ EU-C	*EU CONFIDENTIAL under the Commission Decision No2015/ 444*	
Classified S-UE/ EU-S	*EU SECRET under the Commission Decision No2015/ 444*	

Funded by the European Union

EXECUTIVE SUMMARY

This document reports the WP4 results obtained during the second period of the TEADAL project working towards the main goal of WP4 – addressing the challenges of managing and controlling data sharing flows in the stretched data lake.

In the previous deliverable, D4.1 [4], we have presented the overall control plane concept for TEADAL and proposed mechanisms for realizing the TEADAL data lake on a cloud continuum composed of multiple locations of different types, focusing on resource usage optimization, as well as on creation, deployment, and management of the TEADAL Federated Data Products (FDPs). Initial architectural decisions and technology choices presented in D4.1[4] were verified as part of the integration activities started during the second project period and, in this respect, the current document updates the previous one.

In addition to updating and refining the D4.1 [4], this new deliverable:

- Provides more details on control and management aspects of the TEADAL framework, informed by the evolution of the overall TEADAL architecture (Chapter 8 of D2.3 [3]), by the deeper understanding of TEADAL pilots requirements (Chapters 1 to 7 of D2.3 [3]), as well as by the initial integration activities (D6.2 [7]).
- Presents the technical aspects of integrating the energy saving considerations related to data sharing and data distribution, developed in WP3 (D3.2 [5]), as well as the security and trust features of the TEADAL federation developed in WP5 (D5.2 [6]).
- Reports on new research activities related to simplifying and automating the data sharing processes in the TEADAL federation. Here, the focus is on processes involved in creation, deployment, and management the TEADAL Shared Federated Data Products (sFDPs) and providing the description of new automated sFDP generation tools created using advanced technologies based on generative AI.

Funded by the European Union

TABLE OF CONTENTS

Funded by the European Union

LIST OF FIGURES

Funded by the European Union

LIST OF TABLES

Funded by the European Union

ABBREVIATIONS

AI-DPM	AI-driven Performance Monitoring
AIOps	Artificial Intelligence for IT Operations
ASG	Automatic sFDP Generation
API	Application Programming Interface
CD	Continuous Delivery
CI	Continuous Integration
EDA	Exploratory Data Analysis
FDP	Federated Data Product
GRU	Gated Recurrent Units
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IT	Information Technology
I/O	Input / Output
JWT	JSON Web Token
LLM	Large Language Model
LSTM	Long Short-Term Memory Neural Network
MCC-C	Multi-cloud Computer Compiler
MPC	Multi-Party Computation
MSE	Mean Squared Error
OIDC	OpenID Connect
OPA	Open Policy Agent
PoC	Proof-of-concept
PCA	Principal Component Analysis
RAG	Retrieval-Augmented Generation
RBAC	Role-based access control
REST	REpresentational State Transfer
SFDP	Shared Federated Data Product
TEE	Trusted Execution Environment
TTMs	Tiny Time Mixers
YAML	Yet Another Markup Language
ZKP	Zero-Knowledge Proof

Funded by the European Union

1. INTRODUCTION

One of the main value propositions of the TEADAL project is to create a federated data sharing framework whereby data providers and data consumers can achieve their data usage goals in a more trustworthy, privacy-aware, and energy-efficient way. TEADAL platform achieves this by combining and expanding the Data Lake and Data Mesh concepts, so that the benefits that both businesses and people obtain from data sharing can be achieved across organizational, geographic, and technology boundaries, while minimizing the additional complexity and cost as much as possible.

As a technology work package of the project, WP4 is tasked with the architecture of the TEADAL stretched data managing and sharing platform and, mainly, with the control plane and the integration aspects of the project. The updated state of the overall project architecture is presented in Chapter 8 of D2.3 [3] and the initial architecture of the control plane and the data management components was shared in D4.1 [4]. This deliverable zooms into the architectural details developed in WP4 over the second project period.

This section is an overview of the deliverable structure and its main contributions. Next, Section 2 describes updates on the previous deliverable, Section 3 focuses on newly developed technologies for TEADAL data pipelines automation, Section 4 summarizes details on TEADAL data pipelines optimization, and Section 5 concludes the document outlining the next period plans.

In the reminder of this section, we summarize the contributions of this deliverable: updates to the TEADAL stretched data lake requirements in Subsection 1.1; updates to the TEADAL Control plane in Subsection 1.2; new, gen-AI based, capabilities for automating the TEADAL data sharing processes are summarized in Subsection 1.2; finally, Subsection 1.31.4 summarizes the integration aspects of the advanced energy-conserving capabilities developed during the second TEADAL iteration.

1.1 STRETCHED DATA LAKE AS PART OF TEADAL FEDERATION

The role of Control Plane is central to the TEADAL data sharing capabilities and is required by all the pilots. The purpose of the Control Plane is the deployment and runtime orchestration of data sharing workloads in a stretched data lake, across its different locations. Runtime orchestration mostly consists of managing workload lifecycles, starting with the placement of individual components on suitable infrastructure elements, overseeing component operations, while running, and decommissioning the resources as soon as the work is completed. As was described in D4.1 [4], TEADAL Control Plane is not the sole decision plane of the project. Rather, it works in cooperation with other components of the TEADAL platform by executing actions based on these other components' inputs, to achieve data management goals in the best possible way. For example, in cooperation with the Trust Plane, which might determine if certain components misbehave, the Control Plane is responsible to stop or to quarantine the culprits. Another example is making workload placement decisions based on performance and energy consumption data collected from the nodes and analyzed by the Data Governance components, developed in WP3. Initial Kubernetes based Control Plane architecture was presented in D4.1 [4] and its optimization capabilities were demonstrated as part of the first periodic review, for a single TEADAL location. Since then, the Control Plane requirements were further extended to support each one of the individual TEADAL pilots. As can be seen in D2.3 [3], most pilots include multiple organizations and multiple locations, therefore Control Plane capabilities need to support the Trust Plane, Data Governance, and the TEADAL data pipelines orchestration across the whole federation.
To illustrate the multi-location and multi-organization requirements, we'll use a specific example in Figure 1, where one possible configuration of TEADAL data sharing federation is

Funded by the European Union

presented, with three different organizations, *Organization A*, *Organization B*, and *Organization C*. In this example, each organization has its own personnel managed by its own Identity Management mechanisms, as well as its own internal datasets, and its own IT infrastructure used to host internal data and services, as well as data and services of the TEADAL Federation. Each organization can decide which datasets and infrastructure to share with the federation members, as well as who are the organization's people that might need access to other organizations' data. and what their data access permissions are. According to the TEADAL platform architecture and the data sharing decisions made in WP2, the IT infrastructure is shared in the form of TEADAL Nodes while the data is shared in the form of Federated Data Products (FDPs). This means that each organization installs one or more TEADAL Nodes in one or more locations, either on-prem, cloud, or edge, following the TEADAL GitOps procedures defined in WP6. These TEADAL Nodes join the TEADAL Federation and host a use-case-dependent set of TEADAL services. In addition, these nodes can host one or more internal data sources. Data, which the organization has decided to share is exposed as TEADAL FDPs and is published in the TEADAL Catalog. In the example presented in Figure 1, we show *Organization A* with three TEADAL Nodes one of which is running on the edge; *Organization B* with three TEADAL Nodes, one of which has confidential computing capabilities, and *Organization C* with only one TEADAL Node.

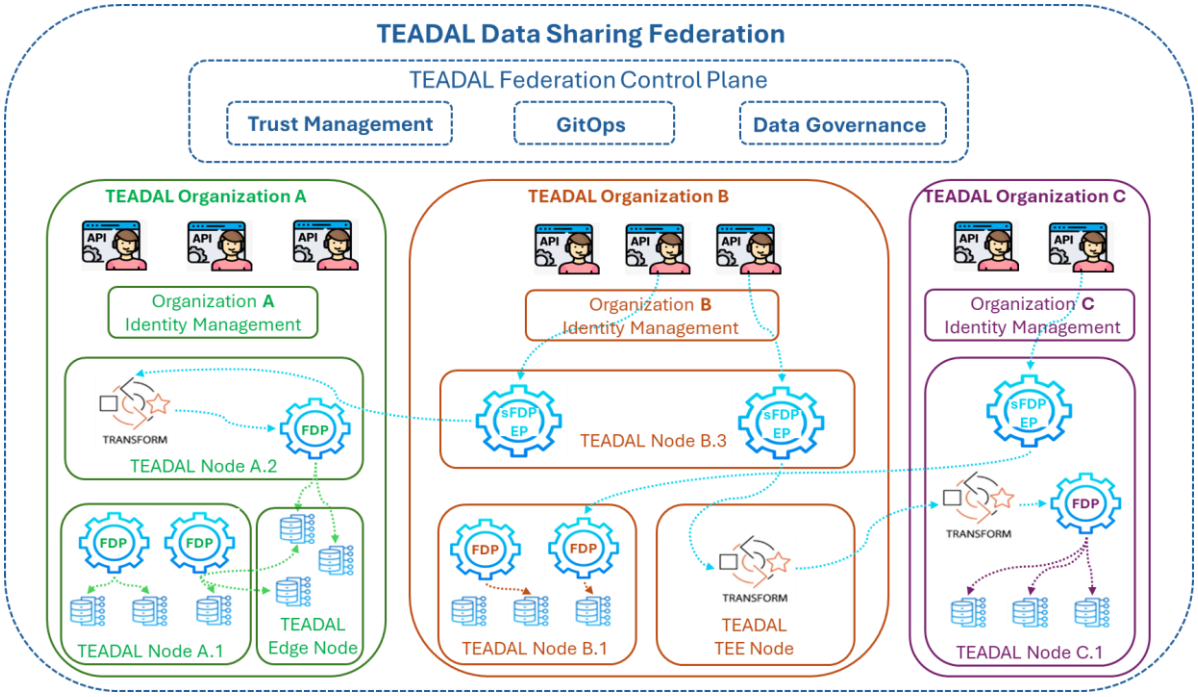

FIGURE 1: TEADAL DATA SHARING FEDERATION EXAMPLE

As described in previous deliverables, the FDPs each organization decided to share are published inside a federation's data sharing catalog where they can be discovered by other organizations. All federation members can discover the FDPs and, if interested, contact the FDP owner to formulate a custom data usage contract. As soon as a contract is agreed between the data producer and data consumer, a new custom data product is created that consumes the FDP APIs and exposes new APIs as defined by the contract. This new data product is a TEADAL Shared Federated Data Product or sFDP. In the example presented in Figure 1, *Organization A* has six internal datasets stored in two different locations and accessible only by *Organization* A personnel according to its internal policies, exposes three FDPs deployed in two different locations, and does not consume other federation members' data; *Organization B* has three internal datasets stored in one location, exposes two FDPs deployed in the same location where the data is stored, and consumes data exposed by *Organization* A and *Organization* C through two different sFDPs; *Organization* C has three

Funded by the European Union

internal datasets stored in one location along with the one FDP and consumes data exposed by *Organization B*. In addition, Figure 1 exemplifies different type of sFDPs: while the sFDP whereby *Organization* C consumes *Organization B's* FDP, is a simple single component service, the two sFDPs used by *Organization* B are more complex and involve several transformation steps implemented as separate service components deployed in different locations to satisfy non-functional TEADAL requirements such as the need to be executed near where the data is stored or to be protected by the advanced confidential computing capabilities like Trusted Execution Environment (TEE), e.g. the TEE Node of *Organization B* in Figure 1.

In addition, Figure 1 shows the TEADAL control plane functions required to operate multiple FDPs and sFDPs across multiple TEADAL Nodes belonging to multiple organizations and installed in different locations. This common functionality is implemented as TEADAL services hosted on selected TEADAL Nodes inside the federation, and includes the TEADAL Catalog, the Data Governance Services developed in WP3, the Trust Management services developed in WP5, and the GitOps services developed in WP6 and extended in WP4 to enable multi-location and multi-cluster operations.

1.2 UPDATED TEADAL CONTROL PLANE

The initial control plane architecture was presented in D4.1[4] , as part of the first phase single-location demonstration at the first project review, featuring the ability to optimize the data processing pipelines based on resource utilization and application metrics. In addition, D4.1[4] formulated the second period goals towards refining and extending the first period achievements in three different directions. Here is a summary of these three goals, together with how they are addressed in the current deliverable:

1. Extending the capabilities for monitoring and optimizations through metadata creation, analysis, and insight extraction. For this goal, in the second period, the team has focused on integrating the energy and performance related monitoring and optimizations developed in WP3 (D3.2[5]). This is described in more details in Subsections 2.1 and 2.2.
2. Demonstrate how the proposed control and management capabilities support multiple locations, simplifying operations of a multi-location data lake. We describe multi-location control flows in Subsection 2.3
3. Demonstrate a data processing pipeline partitioned for the purpose of implementing energy efficiency, data gravity, data friction policies, and to be able to apply the insights obtained from monitoring and observing the data lake and analysing the collected data. An example of such a pipeline is described in Subsection 2.3.2.

1.3 ADVANCE DATE AUTOMATION

One of the major obstacles for leveraging data for cross-boundary analytics is the need to rely on manual steps as part of the data handling and sharing life cycles. During the early phases of the project, a need for several such manual steps were identified. For example, human developers are expected to create configuration files and software required for serving datasets as data products, as well as to create configurations and software required for consuming the data products published for sharing as FDPs and serving their data as sFDPs according to the agreements sealed between the data producer and data consumer. The former case can be referred to as a TEADAL FDP creation pipeline and the latter as a TEADAL FDP to sFDP transformation or a TEADAL sFDP creation pipeline. Both cases rely on traditional software development patterns, and thus, depend on software development teams as part of establishing and maintaining the data sharing processes. This manual approach can potentially be slow, and thus, hinder the velocity of achieving adequate levels of data access and data sharing as part of the TEADAL federation. To cope with this, we have conceived a possibility for employing novel generative AI technologies such as agentic systems to

Funded by the European Union

automate these manual steps whenever possible. Our first realization of this idea is through a concept of Automatic sFDP Generation (ASG) that, given an OpenAPI [12][24] specification of the source FDP and a contract between the data producer and data consumer, generates the deployment artifacts for a new sFDP that complies with the contract and is ready to be deployed and consumed as a shared data product. We start with the simplest cases where the sFDP can be a single component data pipeline acting as a proxy to the source FDP. Such sFDP will be generated as a single REpresentational State Transfer (REST) server that can access the source FDP, perform the required data transformations and serve the result as prescribed in the sFDP OpenAPI specification. For the less trivial cases, where data needs to be subjected to several transformations. under several, possibly location dependent policy rules or unique infrastructure requirements, the ASG componentry will have to be extended to produce multi-component pipelines that, in addition to a data serving REST server, requires the deployment of additional components that realize the required multi-step data transformation chain.

In addition to automating the generation of the sFDP pipelines, ASG is planned to offer automation of other processes such as defining the sFDPs and accessing the sFDPs by the end users, e.g., through convenient end user interfaces, such as a Notebook or another type of dashboard.

Some specific examples of current ASG capabilities are:

1. Given the OpenAPI definition of the source FDP, generate a simple sFDP acting as a connector to a source FDP, like a proxy. Here, the result is a ready-to-be-deployed sFDP server along with its deployment configuration files. No data transformations are performed in this case.
2. Given the OpenAPI definition of the source FDP and the list of the transformations required to be applied to FDP's data before serving it to the sFDP user, generate a data transforming sFDP. In this case, the result is also a ready-to-be-deployed REST server along with its configuration files, but this time the server applies the automatically generated computational chain as an additional step between getting the data from the source FDP and sending it as a reply to the sFDP user request. This automatically generated computational chain implements data transformations provided as input.
3. Given a description of required data transformations based on a producer-consumer contract, provided in human language as a list of the computation steps to be applied to FDP's data before serving it to the sFDP user, interpret the list and generate an OpenAPI specification of the resulting sFDP. This capability augments the sFDP generation capability by eliminating the need to manually construct the OpenAPI specification required for the end users to access data shared as an sFDP.

This new ASG componentry will be further described in Section 3, along with the discussion of the cost and energy considerations involved in applying the generative AI technologies it is based upon, e.g. Large Language Models (LLMs).

1.4 ADVANCE CONTROL CAPABILITIES

As soon as sFDP deployment artifacts are created, either by ASG as described above and in Section 3, or manually by the software development teams, the TEADAL framework takes care of the optimal deployment and serving of the sFDP to its end users through the following advanced capabilities:

1. Provide a REST endpoint and/or a more convenient interface, e.g. dashboard or Notebook, for the end user of the sFDP.

Funded by the European Union

2. Extend the data lake monitoring and add analysis and insight capabilities, creating dynamic metadata that will help optimizing the sFDP deployment, as well as the overall use of the TEADAL data lake.
3. Extend the multi-component sFDP creation pipeline specification to support deploying to multiple locations and demonstrate how the control plane simplifies the management of a multi-location data lake.
4. Demonstrate how sFDP placement is optimized, under the policy constraints, for the purpose of implementing energy efficiency, data gravity, data friction policies, and to be able to apply the insight obtained from monitoring and observing the data lake.
5. Whenever possible, the sFDP deployment optimization goals (in addition to the policy restrictions) will be demonstrated by the resource performance and energy considerations following the methodology put forward in D3.2 [5].

Funded by the European Union

2. UPDATED TEADAL CONTROL PLANE

In this section, we describe how the TEADAL Control Plane has been advanced as part of the overall project architecture and pilot definitions (D2.3 [3]) and following the results in the pilot definition and the integration efforts (D6.2 [7]).

TEADAL control plane, responsible for orchestrating the data lake workloads over the data lake infrastructure, heavily depends on monitoring and collecting various types of metadata to track runtime optimization parameters such as workload performance, infrastructure usage, energy use, etc., as well as on metadata to track privacy and security aspects such as data and API accesses, authentication requests, etc. In the current iteration, TEADAL capabilities for monitoring and optimizations through metadata creation, analysis, and insight extraction were greatly extended, focusing on integrating the energy-based monitoring and optimizations developed in WP3 (D3.2 [5]) as well as the privacy and trust controls developed in WP5 (D5.2 [6]).

We start this section by describing the AI-based approach for TEADAL performance metadata in subsection 2.1, then provide an update on energy related metadata in subsection 2.2, then describe the updated approach for extending the control plane to multi-location cases in subsection 2.3.

2.1 TEADAL PERFORMANCE METADATA: THE AI-BASED APPROACH

Metadata on IT resource utilisation (RAM, Disk Memory usage) is essential for the proactive management and optimization of IT operation performance. In TEADAL, we are using such metadata to detect anomalies and generate predictive insights, which are made available to the Control Plane deployment optimizer to let it define effective strategies to optimise data flows. Our approach is based on Artificial Intelligence for IT Operations (AIOps) and uses metadata to provide valuable insights into system behaviour. Logs, metrics, and event data, collected from multiple IT resources belonging to one or more organisations in a TEADAL federation context, form the core of this metadata.

Building upon the initial feasibility study and the foundational design of AI-driven Performance Monitoring (AI-DPM) in the first iteration as detailed in D4.1[[4]], AI-DPM is being developed following a series of experimental implementations, designed to refine algorithms using metadata from diverse sources incrementally. The succession of the experimental process begins with accessing valuable publicly available metadata from distributed systems, providing a broad foundation for initial algorithm development and testing. The activities carried out during the reporting period focused on the Proof of Concept (PoC) experimentations, which is the first phase of the AI-DPM experiment series. In this document we report the activities of the PoC experimentation along with the activities related to the study and the design of additional experiments, providing the complete experimentation path that we'll follow in the remaining months of the project to reach AI-DPM maturity.

2.1.1 Proof of Concept Experimentation of AI-DPM

During the reporting period, the PoC phase involved using public open data to create a foundation for AI-DPM. The PoC dataset scraped from a Kubernetes cluster using Prometheus [19], includes up to 80 key metrics, and the most reliable data we were able to obtain after several attempts of searching and generating relevant datasets for our experimentation. Specifically, we analysed a range of metrics — 8 CPU, 11 disk, and 47 memory metrics — for the development and testing of the Anomaly Detection and Predictive Analysis AI models. Both

Funded by the European Union

models have been experimented by using different algorithms, as described in the following subsections, although we only present a subset of these results. The detailed report of PoC results, along with the Jupyter Notebook(s) that provide the interactive computational environment that allows us to write and execute code, visualise data, and include explanatory notes, are available in the TEADAL project repository AI-DPM PoC folder[1]. Other than evaluating the AI algorithms, the PoC also concentrated on managing metadata, which involved sourcing credible time-series data, preprocessing it, and performing Exploratory Data Analysis (EDA). Here we present key accomplishments and advancements made during the reporting period in experimental implementations in terms of:

- Metadata sourcing and management methodologies for building AI models,
- Development and testing of core AI models for performance monitoring,
- Results of the PoC experimentation and data lake monitoring implications,
- Directions for integrating new tools and technologies to enhance data sourcing, model development, testing, and performance improvement.

2.1.1.1 Metadata Sourcing and Management in the AI-DPM TEADAL Ecosystem

The AI-DPM system has been designed upon the monitoring ecosystem of TEADAL nodes deployed on Kubernetes, using open-source toolsets to ensure wide-ranging observability, alerting, and visualisation capabilities. This ecosystem primarily consists of Prometheus [19] for metrics collection and storage, *Prometheus Alertmanager*[2] for alert processing and notification, and Grafana [20] for visualisation and dashboarding (as introduced in previous report D4.1 [4], see Figure 2.

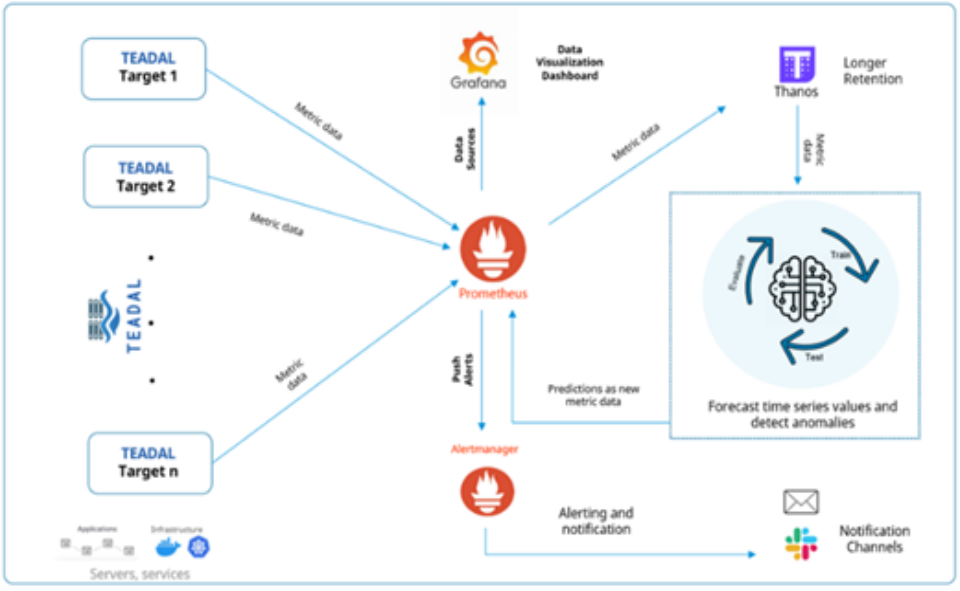

FIGURE 2: AI-DPM ARCHITECTURE FROM D4.1

[1] https://gitlab.teadal.ubiwhere.com/teadal-tech/ai-dpm/-/tree/main/PoC-public-dataset
[2] https://prometheus.io/docs/alerting/latest/alertmanager/

Funded by the European Union

AI-DPM extends this triad by incorporating Thanos [21], which addresses the limitations of Prometheus in terms of long-term storage and global query capabilities. This integration enables AI-DPM to support analytics on historical data, facilitating predictive resource utilisation, and anomaly detection using AI models for monitoring the TEADAL data lake infrastructure. Predictions and anomalies can be sent to Prometheus in order to feed Grafana dashboards with prediction time series and to set up alerts and notifications for specific anomalies. At the current stage of the project, TEADAL components, such as the Control Plane optimizer, are intended to access prediction and anomalies through Rest APIs returning the `[TEADAL node, actual time, time window, predicted value]` predictions array and the `[TEADAL Node, node status, anomaly_flag]` anomalies array.

At the core of the AI-DPM, Prometheus serves as the primary metrics collection and storage system. Its seamless integration with Kubernetes allows the scraping of metrics from various services and components within the TEADAL data lake, providing the raw data necessary for AI-driven analysis. The pull-based model and service discovery mechanisms of Prometheus ensure dynamic and scalable metric collection from the data lake infrastructure. Working with Prometheus, the AlertManager component processes alerts based on predefined rules, which leverages TEADAL prediction and anomalies detection AI-model generated outputs. This integration allows the AI-DPM system to provide intelligent, context-aware alerting that can adapt to the evolving patterns in the data lake's performance based on historical data and contexts.

To support the advanced analytics capabilities of the AI-DPM, Thanos usage extends the functionality of Prometheus by addressing its limitations in long-term storage and global querying. This is crucial for training and running AI models that require extensive historical data. Thanos's components work together to provide unlimited retention of metrics and a global query layer: the Sidecar uploads data to object storage, the Store Gateway serves this data, the Compactor optimises long-term storage through downsampling, and the Querier enables querying across multiple Prometheus instances and object storage. This setup allows the AI-DPM system to maintain and analyse historical metrics over extended periods, enabling trend analysis, anomaly detection, and predictive modelling that form the core of TEADAL's AI-driven performance monitoring.

Grafana serves as the visual interface for the outputs of AI-driven predictive analytics. Its extensive library of visualisation options and plugin architecture allows for the creation of tailored views that can present insights for facilitating quick decision-making and performance optimization.

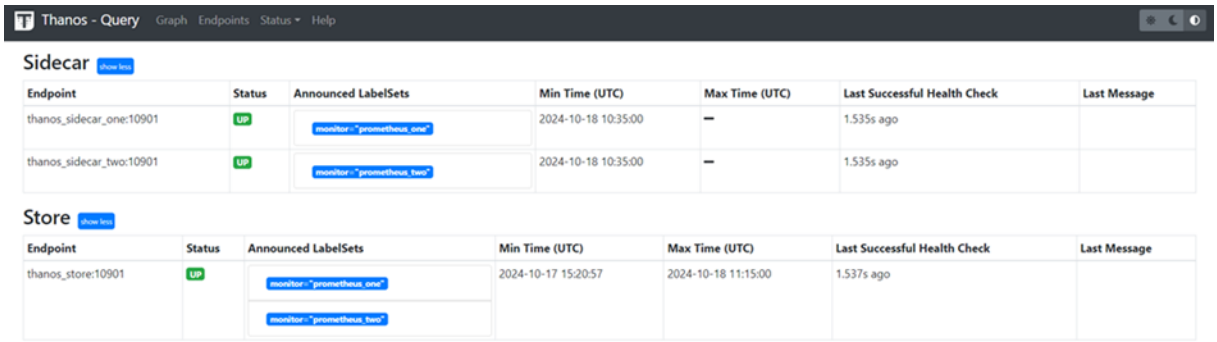

FIGURE 3: THANOS SCREENSHOT IN AI-DPM

As part of developing the AI-DPM system, we conducted initial experiments to set up a basic monitoring infrastructure on a local server. This experimental phase focused on implementing key components of the monitoring ecosystem, specifically installing and configuring Prometheus for metrics collection, Grafana for visualisation, and Thanos for persistence. This

Funded by the European Union

initial experiment provided valuable insights into the foundational aspects of our monitoring system. Figure 3 displays the Thanos dashboard, highlighting key elements like the sidecar endpoint and the store endpoint. It also includes tabs for various functionalities such as "Query" for running Prometheus queries, "Graph" for visualising metrics, and other tabs for managing and interacting with Thanos's extended features, including data retrieval and storage across clusters.

Building on this foundation, we proceeded deeper into the configuration of system-level metrics collection, with a particular focus on CPU, memory, and disk performance. Our experiments explored capturing metrics such as CPU usage percentage, memory utilisation, and disk read/write operations, storing them as time series for trend analysis in Thanos, while testing various Prometheus configurations. Below are examples of basic simplified Prometheus configuration that sets a global 15-second scrape interval and defines a single job for memory usage on localhost:9100. We extended our learning with Thanos with the goal of configuring basic scraping, remote write, and storage functionalities to understand how Thanos integrates with Prometheus and object storage for long-term data retention readily available for training and testing AI algorithms for AI-PDM. The simplified Prometheus and Thanos configuration are presented in Figure 4.

```
global:
  scrape_interval: 15s
scrape_configs:
  - job_name: 'node'
    static_configs:
      -targets: ['localhost:9100']
  - job_name: 'memory_usage'
    metrics_path: '/metrics'
    scrape_interval: 15s
    static_configs:
      - targets: ['localhost:9100']
    metrics_path: '/metrics'
    params:
      collect[]:
        - meminfo

rule_files:
  - 'memory_rules.yml'
```

```
config:
  global:
    scrape_interval: 15s
    evaluation_interval: 15s
  scrape_configs:
    - job_name: 'thanos'
      static_configs:
        - targets: ['thanos-sidecar:10902']
  remote_write:
    -url:"http://thanosreceive:10908/api/v1/receive"
storage:
  tsdb:
    path: /var/thanos/storage
    retention: 24h
  objstore:
  type: S3
  config:
    bucket: "thanos"
    endpoint: "endpoint_for_thanos"
    access_key: "ACCESS_KEY"
    secret_key: "SECRET_KEY"
```

FIGURE 4: THANOS CONFIGURATION IN AI-DPM

Through these experiments, we gained valuable insights into effectively collecting, managing and storing metadata ready with sufficiently aggregated and persisted data, for training and testing AI algorithms, which is crucial for the next AI-DPM series of experiments.

2.1.1.2 AI_DPM PoC Experimentation with Public Dataset

The dataset used for PoC experimentation uses OpenTSBD [22] notation, which is the format of the Prometheus metrics scraped from specified hosts/targets from any distributed systems, as will be the case in the TEADAL data lake. The data is time-series in nature encompassing CPU usage, memory usage, and disk I/O, among others. This public dataset, collected from a cluster of 332 compute nodes over a five-day period, provided the PoC experimentation data for predictive insights and anomaly detection to investigate the system performance. “Performing” in this case can relate to different aspects — availability, level of stress, under-utilisation of resources, etc.

Funded by the European Union

In the contexts of AIOps, metrics are numeric measures over intervals of time of the internal states of the system, in this case a distributed system or the TEADAL data lake. Thus, metrics are the main piece of information in the monitoring system and more so for AI-DPM. A recorded metric can be described as a combination of two elements, identification, and sample. For example, the Prometheus metric is composed of Identification (with Metric Name and Labels) and Sample (with Timestamp and Value). Examples of the key metrics scraped by Prometheus include system metrics like

- `node_cpu_seconds_total` for CPU usage
- `node_memory_MemAvailable_bytes` for available memory
- `node_disk_io_time_seconds_total` for disk I/O time.

For the PoC, we used all three (CPU, memory and disk I/O) datasets for developing and testing core AI models. Figure 5 shows an example of memory metrics (used node memory and free node memory) status data in a five-day window in the `332` node-cluster that was used for PoC experimentation.

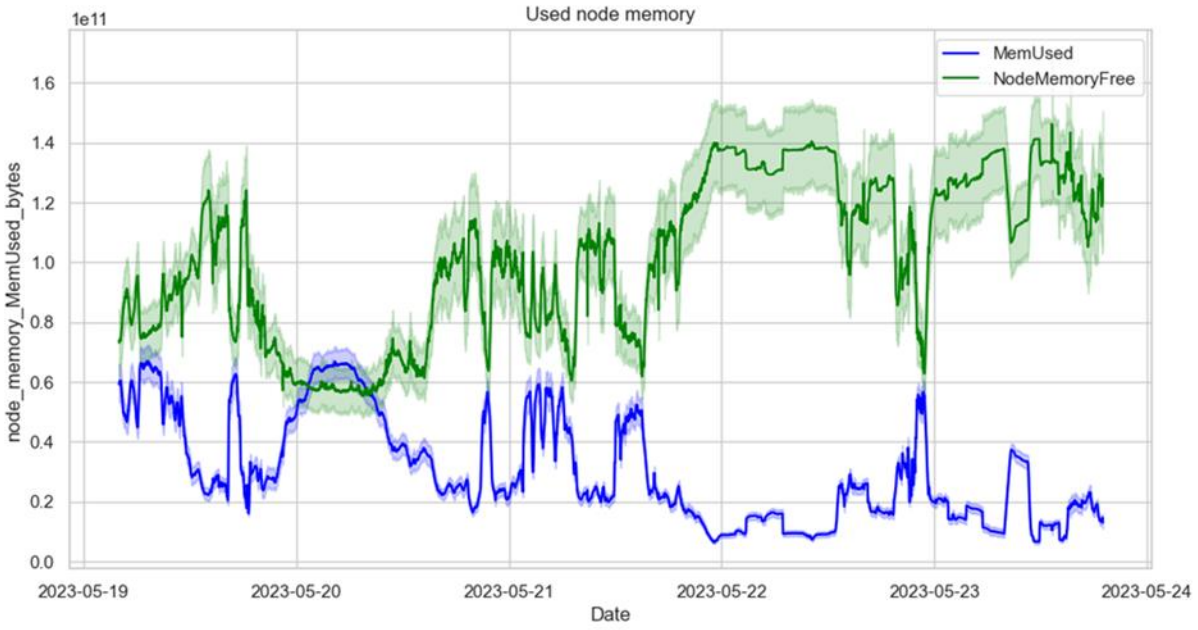

FIGURE 5: MEMORY METRICS STATUS DATA IN AI-DPM

2.1.1.3 Predictive Analysis Models

The AI-driven platform monitoring approach leverages Machine Learning (ML) to enhance traditional monitoring of infrastructure, applications, and services. Prediction metadata can offer insights into expected infrastructure utilisation (e.g., CPU, Memory, or Disk I/O) for definite future time windows. In the context of the TEADAL project, this generated new metadata will be available to the Control Plane as more specific metadata for enabling system optimization.

Funded by the European Union

The PoC experiment has explored four classical predictive analysis algorithms including, Long Short-Term Memory Neural Network (LSTM) [8], Gated Recurrent Units (GRU)[3] [9], IBM's Tiny Time Mixers (TTMs) [10], and Facebook's Prophet (Prophet) [23]. These models were applied to a five-day dataset of node memory usage collected from a distributed cluster computing environment. The experiment utilized data from the first two days for training and the third day for testing, with a temporal window considering 60 minutes of historical data to predict the next 20 minutes of available node memory.

Performance evaluation based on Mean Squared Error (MSE) as presented in Table 1 revealed that GRU performed best with an MSE of 0.120060, closely followed by LSTM at 0.123420. TTMs-r1 from IBM showed moderate performance with an MSE of 0.331747, while Facebook's Prophet had the highest MSE at 0.420203. These results suggest that recurrent neural network architectures (GRU and LSTM) were more effective in capturing the temporal patterns and dependencies in the node memory usage data compared to the other approaches. In other cases, TTMs-r1 outperformed other models, such as in predicting the node disks write time (seconds) metric, with GRU as the second-best performer.

TABLE 1 : PREDICTIVE ANALYSIS MODEL PERFORMANCE

Predictive analysis AI models	**Mean Squared Error (MSE)**	
	Node memory used (bytes)	Node disks write time (seconds)
LSTM	0.123420	0.158195
Prophet	0.420203	1.483038
GRU	0.120060	0.131413
TTMs – r1 (IBM)	0.331747	0.010300

Prediction visualisation plots for the investigated four models are provided in the TEADAL project repository AI-DPM PoC folder. In Figure 6, memory usage predictions for different node instances using the better performing model, that is LSTM. In the figure, historical data, actual values, and model predictions are shown for each instance. While the predictions align well with actual data for some instances, they show discrepancies for others. Overall, the results suggest the need for further model testing and potential model ensemble approaches. All code snippets and notebooks are available in the TEADAL project repository.

[3] GitHub Selective Update GRU

Funded by the European Union

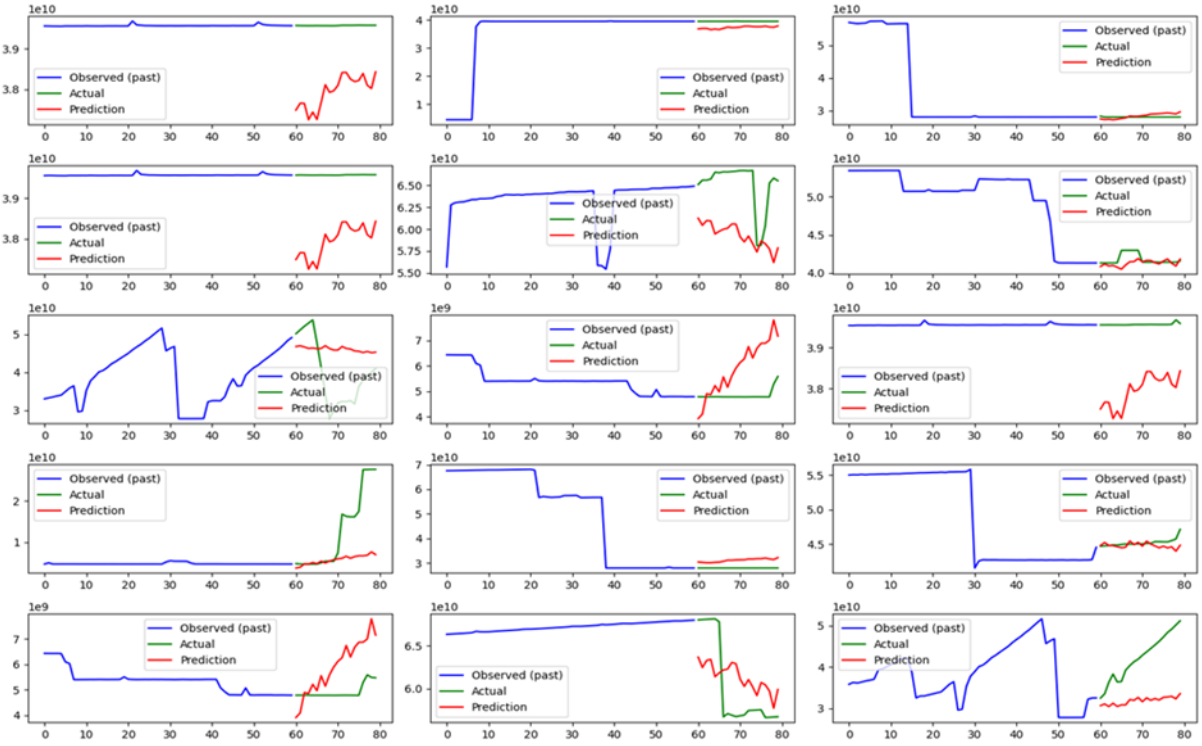

FIGURE 6: MEMORY USE PREDICTIONS WITH LSTM

2.1.1.4 Anomaly Detection Models

The AI-driven platform monitoring approach can also detect anomalies, facilitating proactive management and optimization of resources. For example, anomaly flag metadata can identify stressed or under-utilised resources.

The anomaly detection experiment, utilising a combination of statistical threshold methods and K-means clustering, analysed available node memory time-series data from a 332-node cluster. Focusing on 11 memory-related features, the analysis identified 4 nodes (1.2% of the cluster) exhibiting anomalous characteristics, with 26,624 anomalies detected out of 2,203,252 data points, representing a 1.21% anomaly rate. These findings have significant implications for cluster management, potentially serving as an early warning system for memory-related issues, enabling better resource allocation and load balancing, and guiding predictive maintenance efforts. At this part of the project, despite not having real TEADAL data for validation, the PoC can provide valuable insights and a foundation for further refinement of anomaly detection techniques in AI-DPM in TEADAL's monitoring ecosystem. Notebooks with code snippets and more results are made available in the TEADAL project repository AI-DPM folder.

Figure 7 below shows the distribution of five-day memory-related time series data for some of the metrics recorded across 332 nodes. It highlights the range and variation of each metric, helping to understand the overall patterns and deviations in the dataset and providing insights into system operating conditions.

Funded by the European Union

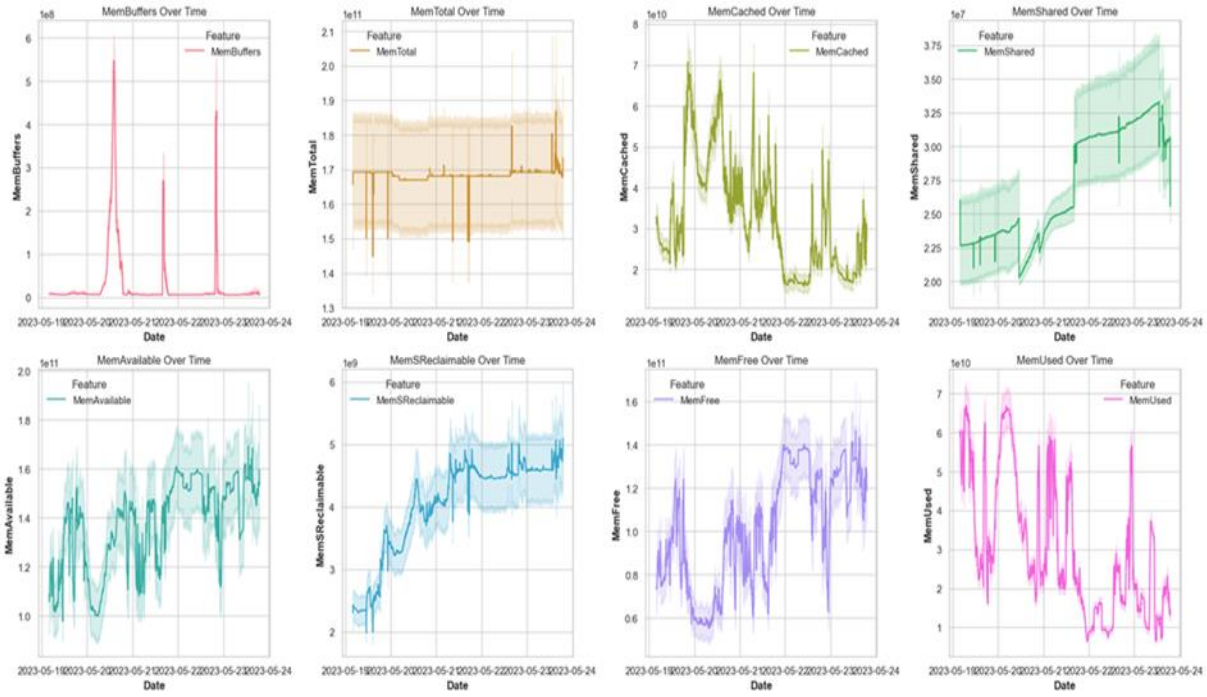

FIGURE 7: MEMORY-RELATED TIME-SERIES DATA

The left plot in Figure 8, generated using Principal Component Analysis (PCA), displays data projected onto the first principal components. Each dote indicates a node instance with its timestamp recorded and it is labeled as outlier (1, red colored) or non-outliers (0, black colored) based on KMeans clustering. The plot moderately separates clusters, with outliers standing apart from the main group, aiding in anomaly detection. The right-side plot, a node instance with its timestamp recorded for the entire five-day window and the specific anomaly points detected using the statistical method.

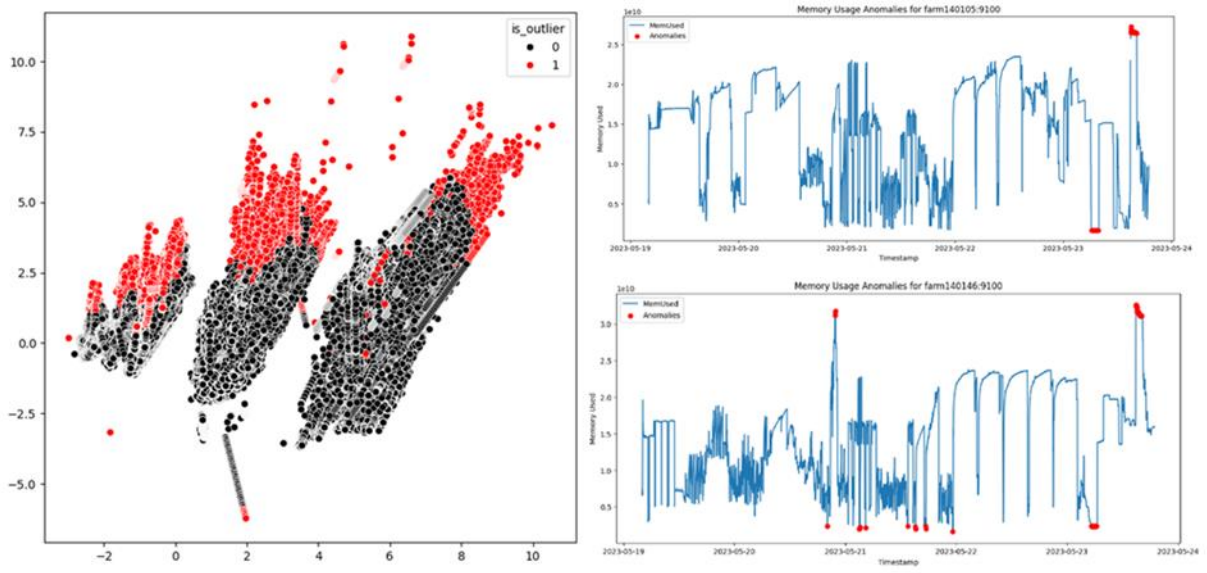

FIGURE 8: MEMORY RESULTS OF KMEANS CLUSTERING

2.1.1.5 Anomaly Detection Models

Our initial PoC experimentation with public metadata provided valuable insights and revealed several important implications for the next phase AI-DPM development. First, realising four

Funded by the European Union

different predictive models for this PoC offers several key advantages. The development and testing of models enabled a comparison of algorithms and the establishment of a performance baseline for benchmarking. The approach also demonstrates flexibility in handling different types of metrics beyond just memory usage, providing insights into each model's strengths under specific conditions. This multi-model strategy lays the groundwork for potential ensemble methods, allows for scalability assessment, and offers adaptability to changing environments. Ultimately, this diverse modelling approach enhances the ability to select and refine the most effective predictive techniques for complex, real-world scenarios involving multiple metrics and evolving system characteristics of TEADAL.

Nevertheless, the results should be carefully examined for implications in TEADAL data lake AI-DPM. The system difference - the PoC was conducted on a dataset sourced from a system that might differ from the target TEADAL data lake environment in terms of the microservices running and the patterns observed in the data. The PoC data - the fact that the PoC data is not labelled means that it lacked ground truth data for validation of anomalies, moreover, the data used can enable only short training and testing periods, which may not capture long-term trends or seasonal patterns resource use in the infrastructure.

Overall, the results of the PoC provided important future directions that could involve extending the study to longer periods, incorporating multiple performance metrics, and exploring ensemble methods that combine the strengths of different models. Additionally, conducting experiments on datasets resembling the TEADAL data lake environment or the TEADAL environment itself would be crucial for validating the models' effectiveness. Further investigation into the scalability of these models especially considering the specific characteristics of the TEADAL environment and integrating the results of algorithms into the TEADAL monitoring ecosystem would be required for real-world applications. Details of this important future direction is provided in the following section.

2.1.2 AI_DPM Full Experimentation Design

While progressing with AI-DPM experimentations, we started exploring approaches to enhance the data-sourcing strategies and optimise model development and testing approaches. In this regard, the two key directions we considered for future work are the creation of a dedicated experimentation environment and adding Large Language Models (LLMs) capabilities into our forecasting and anomaly detection AI model stacks.

2.1.2.1 µBench for Creating an AI-DPM Experimentation Environment

The first PoC experimentation with public metadata highlighted the need for more specialised simulated metadata. In order to address this need, we started to design a dedicated experimentation environment and to seek for a suitable metadata generation tool.

Our researchers have discovered and introduced to the experimentation process the µBench[4] [11], an open-source software that emulates real-world Kubernetes cluster scenarios. It is distributed under the BSD 4-Clause License and includes software developed by University of Rome Tor Vergata and its contributors.

[4] https://github.com/mSvcBench/muBench

Funded by the European Union

µBench is a tool designed to assess the performance of microservices within a Kubernetes environment. It allows users to simulate the behaviour of an application composed of multiple microservices (such as web services or databases) by performing load tests and collecting metrics like CPU, memory, network, and disk space usage. µBench provides a comprehensive monitoring framework consisting of Prometheus, Grafana, Istio, Kiali, and Jaeger through which to observe and display the performance of the applications.

In the context of PoC experimentation, µBench enables the simulation of a distributed microservice environment that mirrors key aspects of a TEADAL data lake's infrastructure, such as computing, storage, and data processing components. By creating mock services and running load tests, µBench can generate key performance metrics under various workloads, allowing experimentation with real time data before the TEADAL testbeds' data lake infrastructure is fully operational. Moreover, µBench simplifies the experimentation process by facilitating the configuration of Prometheus, Grafana, and Thanos, while managing the end-to-end data flow for the AI models.

To improve the initial PoC, we plan to integrate µBench into a dedicated experimentation environment with two Virtual Machines (VMs) provided by ALMAVIVA.

µBench is crucial for creating

1. *TEADAL-Like Workload Simulation*: µBench will be used to generate synthetic workloads that closely mimic the characteristics of the TEADAL data lake environment, including its unique microservices and data patterns.
2. *Controlled Anomaly Injection*: µBench can be used to systematically inject anomalies and resource stress into the test environment, providing a much-needed dataset for training and evaluating core AI models.
3. *Benchmark for Model Comparison*: by establishing a consistent testbed with µBench, it is possible to compare more accurately the different versions of core AI models and assess improvements over time.

2.1.2.2 Augmenting Classical AI Models with LLMs

While proceeding with testing classical AI models, we started investigations to explore the potential of adding Large Language Models (LLMs) into our forecasting and anomaly detection classical model stacks.

This novel approach could offer several benefits. LLMs are useful for monitoring both on-premises and cloud-native resource performance monitoring. In addition, the use of LLMs alongside classical approaches is essential to leverage their advanced capabilities in handling complex, high-dimensional data. LLMs excel in capturing intricate temporal dependencies and correlations, reducing the need for extensive manual feature engineering and potentially increasing accuracy and adaptability. By conducting a PoC with LLM model working on real data generated in a dedicated experimentation environment, we will evaluate their performance in real-world scenarios, compare it directly with our existing approaches, and assess improvements in prediction accuracy, anomaly detection precision, benchmarking and overall system scalability.

The combined approach will ensure that we are utilising the most robust and effective and up-to-date tools available for monitoring the performance of data lake infrastructure.

2.1.2.3 The Continuum of AI-DPM Experimentation

Funded by the European Union

Overall, the experimentation design aims to ensure that AI-DPM will be robustly prepared and optimised for the full implementation with TEADAL's comprehensive runtime metadata. The schematic representation of the steps composing the full AI-DPM experimentation is presented in Figure 9.

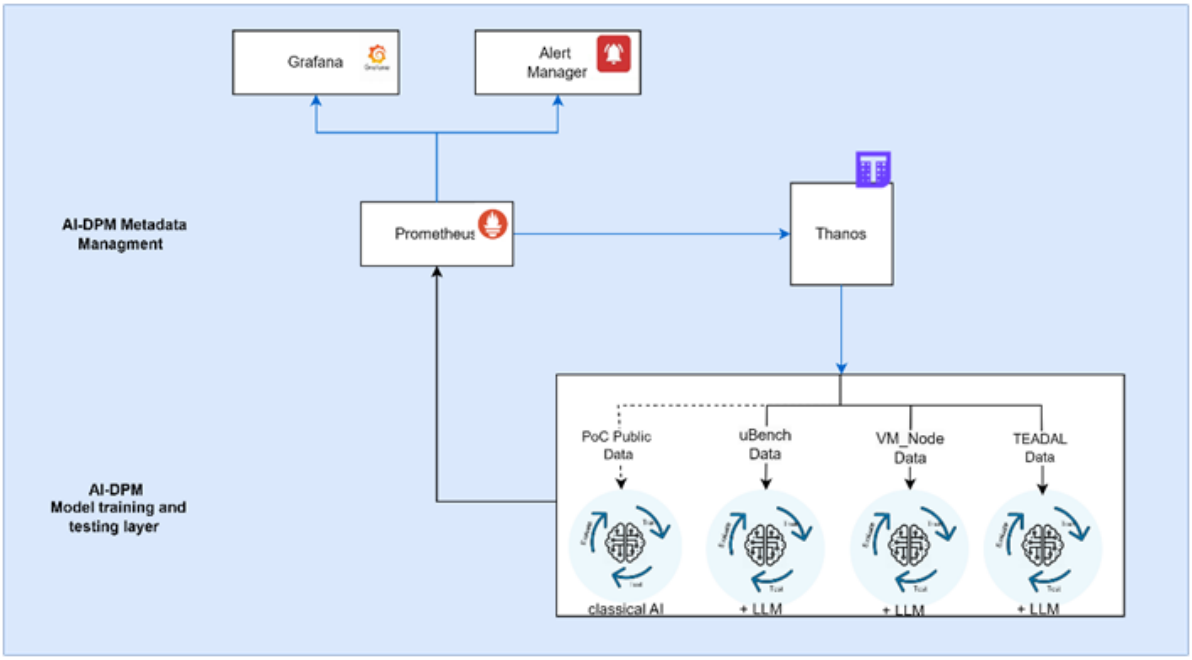

FIGURE 9: AI-DPM EXPERIMENTATION SETUP

In the figure, the AI-DPM training and testing layer is plotted, representing four progressive improvements: *PoC public*, *µBench* (on local machine), *VM_node* (simulated double-node experimentation environment), and *TEADAL Data* (real data from TEADAL Pilots in their final topology). Data is managed primarily through Prometheus and Thanos in the metadata management layer, except for the PoC experiment using public data (indicated by a broken line). Classical AI and LLMs will be used for training, testing, and evaluating AI models for all datasets, except the PoC experiment that used public data (labelled as classical AI in the PoC data and +LLM for the others to indicate the argumentation of LLMs to classical AI models).

Throughout this process, starting from the PoC, multiple algorithms focused on anomaly detection and predictive analysis on multiple sets of metrics are developed and tested. The Large Language Models (LLMs) approach seamlessly adapts the AI classical approach to the increasing specificity and complexity of the metadata.

Further experimentation advancement is achieved by incorporating data simulated on a dedicated experimentation environment alongside µBench with microservices related to TEADAL. At the far end of the metadata source continuum is TEADAL metadata, which represents actual, domain-specific data lake metadata, the target for applying AI-DPM. This structured experimental continuum enables the systematic development and enhancement of AI-DPM, with each stage bringing increased relevance and complexity to its intended use cases.

2.1.3 AI-DPM Next Steps

As we progress with successions of AI-DPM experimentations, the next steps involve setting up a two-node cluster using virtual machines, with monitoring stacks connected to TEADAL-like microservices initialised on µBench. The AI-DPM data collection and management

Funded by the European Union

pipeline, using Prometheus and Thanos, will be implemented to store datasets from this system. Additionally, anomalies will be injected, and stresses of resources will be applied to generate ground truth labelled data for training, testing and evaluating the AI models. Furthermore, LLMs will be incorporated into the stacks of classical AI models for comparative analysis and benchmarking. These efforts aim to improve the AI-DPM system for more efficient data lake management and monitoring within TEADAL.

2.2 TEADAL METADATA FOR ENERGY

When considering the energy efficiency or sustainability of data pipelines in general, we need to be able to estimate the energy consumption and resulting emissions for a specific location. For this, the performance measurements can be combined with metadata about execution locations. Data sources such as Electricity Maps[5] can be used to obtain current carbon emissions based on data center location. This can be integrated into the current TEADAL approach, as the data pipelines also need location information to ensure data access and processing policies. Hence, combining performance measurements and estimations with current carbon emissions and electricity prices enables the estimation of the execution cost for a given pipeline, which we can consider when optimizing pipeline execution and placement.

It should be stated that energy considerations (as well as performance considerations) are an optimization goal for that respect, while policy considerations (e.g. country limitation on data) are hard constraints that should be respected in any placement solution.

To make node location information accessible to energy optimization algorithms in a federated environment, we will leverage Advocate. Advocate, presented in Figure 10, is a novel agent-based system designed to generate attested evidence for applications in the cloud-based federated environment of TEADAL.

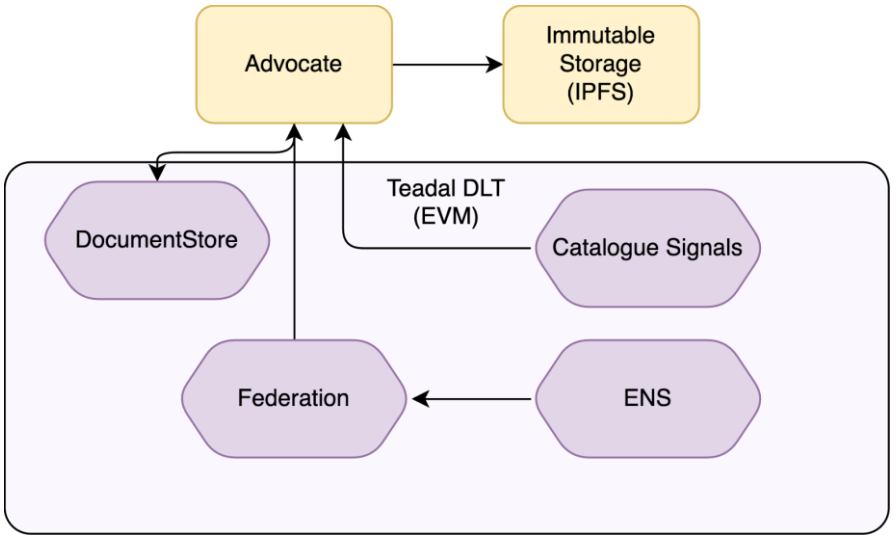

FIGURE 10: SMART CONTRACT INITIALIZATION AND CONNECTION BY ADVOCATE

[5] https://static.electricitymaps.com/api/docs/index.html

Funded by the European Union

By integrating with existing infrastructure tools, such as Kubernetes and observability services, Advocate captures, authenticates, and stores evidence in a tamper-resistant manner using blockchain technology. Advocate deploys a smart contract on the federation's blockchain, to store generated evidence by smart contract which are about events that are published by the smart contract including evidence related to membership management. These contracts are initialized with a predefined set of federation members during the setup phase. By requiring administrators to include node location data as part of smart contract initialization, we ensure that infrastructure location metadata is embedded within the federation.

Node locations are also stored as claims tied to smart contract events within Advocate, ensuring that location metadata is auditable and accessible. This setup enables optimization algorithms to use the metadata for pipeline placement, aligning with the Electricity Map to enhance energy efficiency.

2.3 MULTI-LOCATION CONTROL FLOWS IN TEADAL FEDERATION

In the previous deliverable of this work package, the preliminary control plane architecture was presented for managing workloads in the TEADAL stretched data lake which is a data lake consisting of datasets and data products belonging to a single organization deployed in several different geographically distributed locations. In this stage of the project, control plane concepts need to be extended to support TEADAL federation with multiple participating organizations, each having its own stretched data lake deployed over multiple locations. As far as a single organization's stretched data lake is concerned, the focus was on storing datasets and on hosting data processing pipelines with steps such as ingestion, curation, transformation, etc., up to the ready-to-be-shared FDPs. For a multi-organizational data lake, we are mostly concerned with managing the access to FDPs which, according to the TEADAL architecture, is implemented through a concept of sFDP, an additional data product created to serve the needs of a specific data consumer inside the federation.

The introduction of the sFDP concept adds a layer of abstraction and control over how data is shared and consumed between organizations, featuring the following key aspects:

- Decoupling Access to the data: an sFDP allows the data consumer organization to act as a proxy or intermediary between its data users and the data products offered by data provider organizations. This creates a layer of control for the data consumer organization, enabling them to manage how the data is used internally without directly exposing users to data providers' FDPs.
- Further negotiation of access terms: although FDPs expose data according to policies defined for them by the data provider, the ssFDP concept allow the data consumer and the data provider organizations to negotiate additional specific terms, such as:
  - Which subset of the dataset should be accessible by which users in the data consumer organization.
  - What transformations need to be applied (e.g., anonymization, removing private fields) and what are infrastructure restrictions and requirements for applying them (e.g., data should only be stored on compliant media and in certain locations).
  - What limits should be imposed on data usage (e.g., rate-limiting, allowed operations) for this specific provider-consumer contract.
- Flexibility in sFDP creation: an sFDP can be created by a developer from a data consumer organization, by a developer from the data provider organization, or by a third-party

Funded by the European Union

developer, with no restrictions; in addition, there are opportunities for automatic generation of the sFDP code.

- Additional optimization opportunities: in addition to transformation steps mandated by the sFDP contract, the sFDP pipeline can include additional steps, such as transforming, compressing, filtering, aggregating, or caching, to further refine or optimize the dataset or the overall system performance.

With all the beneficial features, the sFDP concept, together with the extension of the stretched data lake to become multi-organizational or federated, have raised new challenges and considerations for implementing the control plane. For example, deployment of the sFDP data pipeline allows for more flexibility and, at the same time, is subject to more restrictions than the FDP data pipeline which is typically confined to its organization's boundaries. The sFDP pipeline can be deployed partially in the data provider locations, partially in the data consumer locations, or in the locations of other federation members. In addition to being the deployment and runtime data sharing platform, TEADAL aims to be a platform for developing data products and the related data processing and data sharing pipelines. This adds additional requirements to the control plane, mostly in the areas of access control for developers (trust plane) and the integration with the development platforms (GitOps).

To be able to deploy FDP pipelines in a stretched data lake potentially containing several TEADAL Nodes (each running in Kubernetes cluster), D4.1 [4] has evaluated the use of the Kubestellar project for unified workload management across clusters. Although this approach could be possibly extended to support a TEADAL federation with multiple stretched data lakes belonging to different organizations, we have decided to re-open this decision and evaluate additional options. In what follows, we present the technology candidates for managing workloads over a multi-cluster multi-organizational infrastructure and describe their pros and cons. The decision regarding what mechanism will be selected for integration as part of TEADAL pilots will be obtained after additional planned research and experimentation and will be reported in the final deliverable of the work package, D4.3.

2.3.1 Open-Source Software for Multi-cluster Management

To deploy TEADAL services, in particular the FDP and the sFDP pipelines, across multiple Kubernetes clusters, we need tools to allow a group of clusters managed by different organizations to work together in a coherent manner. There are multiple such tools, typically referred to as federation or multi-cluster management tools. For example, KubeFed [28] (Kubernetes Federation) is probably the earliest of systems proposed for managing multiple Kubernetes clusters as if they were a single entity. Since its first inception in 2018 as KubeFed v1 that allowed deploying k8s services over a set of clusters, it has evolved into KubeFed v2 that can handle multiple types of shared resources and is still a viable option for managing multi-cluster environments, although the project was officially sunset by the Multicluster Special Interest Group in 2022. Multiple other options, both commercial and open, have emerged offering different solutions with different set of features. Here we examine some of the currently existing solutions for their applicability to TEADAL needs.

2.3.1.1 Solutions for Kubernetes Cluster Federation

The first group of solutions roughly follows the direction of KubeFed and offer cluster federation in one way or another. Note that in this section we speak of federating k8s clusters which is different from the notion of TEADAL Federation that federates organizations willing to share IT resources and data between them. In Kubernetes, cluster federation enables the loose coupling of multiple independent Kubernetes clusters, making it possible to aggregate the resources and services of multiple clusters into a single, logical entity while keeping each cluster isolated and independent. Most cluster federation solutions are architected with a designated *host* cluster that manages deployments and propagates changes to a set of

Funded by the European Union

participating clusters; an *interface*, be it an API, a CLI, or sometimes a GUI, allowing operators to issue commands across the whole federation; and a set *configurations* that describe the participating clusters and the resources shared among them and provide baseline parameters for arranging cross cluster communications, such as access controls, network access, etc.

The most cited benefits of cluster federation solutions are:

- Single pane of glass for managing state and configuration across clusters
- Cross-cluster discovery: cluster federation solutions typically take care of configuring cross cluster communications, e.g., through load balancers and DNS entries
- Resource syncing across clusters: most solutions offer synchronization of state between clusters which is very useful when it is needed to maintain the same deployment across multiple clusters

At the same time, cluster federation solutions often have the following drawbacks:

- Higher network bandwidth and associated costs: typically, the control plane component deployed in the host cluster monitors all clusters to ensure they maintain the expected current state. This can be especially significant for a large number of clusters installed in multiple locations with non-uniform network availability and capacity.
- Relying on a centralized control plane component can limit cluster independence and cause problems as the control plane in the host cluster experiences faults.

Here we describe some of the solutions under this type:

- **KubeFed**, is a tool for coordinating the configuration of multiple clusters in Kubernetes allowing users to determine which clusters KubeFed will manage, and what their configuration looks like, all from a single group of APIs in the hosting cluster. Users manage Kubernetes resources across multiple clusters using federated types such as FederatedDeployment, FederatedReplicaSet, FederatedSecret, etc. KubeFed system synchronizes these federated objects' state across clusters using push-based model. Even if discontinued, it is still employed in operations and integrated with many modern open-source tools [29][30].
- **Open Cluster Management (OCM)** [31] is a framework to orchestrate Kubernetes functionalities across multiple clusters and cloud providers. This includes built-in functionalities, such as cluster inventory and workload placement and allows extensions to orchestrate any possible Kubernetes-compatible objects and behaviours, such as governance and observability. This modularity and extensibility make OCM popular as a basis for other multi-cluster solutions that create additional capabilities on top of it, e.g. Kubestellar. Also, OCM ccollaborates with popular Kubernetes ecosystem projects, in particular ArgoCD and Istio, easing their application over multiple clusters.
- **KubeAdmiral** [34][35] is a multi-cluster management system for Kubernetes, developed from KubeFed v2. It extends the KubeFed v2 API, providing compatibility with the Kubernetes native API and more powerful resource management capabilities.
- **Karmada (Kubernetes Armada)** [32] is a Kubernetes management system for running cloud-native applications across multiple Kubernetes clusters and clouds, offering turnkey automation for multi-cluster application management in multi-cloud and hybrid cloud scenarios. Kamada features clean integration with ArgoCD and supports both push and pull cluster management modes.
- **Admiralty** (*formerly multicluster-scheduler)* [33] is a system of Kubernetes controllers that intelligently schedules workloads across clusters. It is simple to use and simple to integrate with other tools. Admiralty integrates with Kubernetes at the pod level and uses standard Kubernetes API resources, like Deployments to globalize micro-services, or

Funded by the European Union

Jobs to spread batch workloads, even third-party custom resources. Admiralty supports centralized and decentralized cluster topologies, within and across trust domains.

- **KubeStellar** [36] is a flexible solution for challenges associated with multi-cluster configuration management for edge, multi-cloud, and hybrid cloud. KubeStellar was incepted in IBM research and was accepted to CNCF on December 19, 2023 at the Sandbox maturity level. Kubestellar is designed to centrally apply Kubernetes resources for selective deployment across multiple clusters; to use standard Kubernetes-native deployment tools, in particular kubectl, Kustomize, and ArgoCD; and to make disconnected cluster operation possible. All these features contributed to us selecting Kubestellar for the initial TEADAL control plane implementation. Since then, the project went through major re-architecture, to cater to the enterprise grade goals of supporting high diversity between participating clusters [37]. This made Kubestellar more modular and extensible on the one hand, while on the other hand, the project is now more heavyweight and might be too opinionated for a research project such as TEADAL. Kubestellar is now based on OCM and KubeFlex projects.

2.3.1.2 Solutions for Distributed Management of Multiple Kubernetes Cluster

Cluster federation solutions are suitable the TEADAL control plane as they can allow us to deploy FDP and sFDP pipelines across clusters, to apply policies, and to handle communication between clusters. On the other hand, this solution can incur high computational and network overheads and might be difficult to integrate with custom TEADAL features such as TEADAL catalogue and metadata, Trust Plane, etc. Here, we consider possible alternatives to the full stack cluster federation solutions presented in 2.2.1.1 that might be simpler to adopt and to customize for the needs of the TEDAL project and its specific pilot use cases.

- OCM [31]. First, we can directly use the Open Cluster Management (OCM) [31] and not more complex solutions created on top of it such as Kubestellar. On the one hand, we might need to rewrite some of the functionality, while on the other hand, we will have more flexibility on deciding how to specify and manage TEADAL specific objects and processes, such as data products, data pipelines, policies, GitOps, and trust management.
- Cluster API (CAPI) [38]. Cluster API is a Kubernetes sub-project focused on providing declarative APIs and tooling to simplify provisioning, upgrading, and operating multiple Kubernetes clusters. Cluster API was initially developed by the Kubernetes Special Interest Group (SIG) Cluster Lifecycle, to allow management of k8s infrastructure in a way k8s is used to manage cloud-native workloads and is now a Kubernetes sub-project. While originally created for managing the infrastructure in an infrastructure-as-code way, CAPI is capable of bootstrapping workloads across multiple clusters through integration with GitOps tools such as ArgoCD [39]. This makes CAPI an eligible candidate for TEADAL, at least for some pilots, e.g. where one organization is responsible for the TEADAL Federation infrastructure and offers TEADAL services to its clients.
- GitOps-based Deployment with ArgoCD. Argo CD is a popular GitOps tool for managing Kubernetes deployments. It is already used in TEADAL to streamline the deployment of TEADAL Nodes on k8s clusters and this usage can be extended to the deployment and versioning of FDP and sFDP pipelines across the TEADAL Federation. In certain configurations, ArgoCD can synchronize deployments across multiple clusters. Integrating with TEADAL Node repositories in GitLab, it can allow managing both the infrastructure and the data pipeline configurations in a version-controlled, declarative manner.

2.3.1.3 An outline for a GitOps based Federation Management Solution

Here, we briefly outline the emerging idea to base the TEADAL orchestration as much as possible on GitOps, to allow for declarative management of infrastructure, workloads, and

Funded by the European Union

configurations at all levels of the TEADAL stack. This is an initial outline, to be experimented with in the last project period and to be fully described, if accepted, in D4.3

The key new components of the solution are:

- **Argo CD for GitOps-Based Deployment**. TEADAL already employs ArgoCD to automate the synchronization of Customize manifests between TEADAL Node clusters and their backup repositories in GitLab. Each TEADAL Node has its own repository in GitLab. This ArgoCD setup can be extended to install FDP and sFDP components through pushing changes to their configuration manifests to the correct Kubernetes clusters.
- **GitLab CI/CD (Actions and Scripts)**. GitLab CI/CD can automate workflows, such as managing infrastructure, and handling configuration changes. For TEADAL, this can be used to ensure deployment of the FDP and sFDP components to the correct TEADAL Nodes. When changes occur, e.g. new FDP or sFDP configurations are pushed to the special GitLab repository, GitLab pipelines can trigger actions like creating or updating the related manifests, applying configurations, or rolling out changes across multiple clusters by pushing these manifests to the correct clusters. Depending on use case, this special. GitLab repository can be designed to backup specific TEADAL pipelines (a repository per pipeline instance) or TEADAL organizations (a repository per organization to back up all the workloads running on its behalf), or the whole TEADAL Federation (a single repository to store all the configuration of this federation). In addition, GitLab actions can automate cluster registration, service account creation, and even manage Kubernetes infrastructure with help of other tools such as CAPI.

In addition, the solution will require integration with TEADAL Catalogue and existence of a component to make placement decisions that map policy requirements for running certain pipeline components to the most suitable TEADAL Clusters and can rely on workflow engines, e.g. Argo Workflows [40], for deploying complex multi-step pipelines.

Here one possible way this solution will work for sFDP deployment:

- Each time new sFDP deployment artifacts are created, these assets, in particular the pipeline deployment specification with all the metadata and policy labels, are pushed to the special GitLab repository responsible for that pipeline, organization, or federation, depending on use case, as described above.
- A GitLab action will be triggered to invoke the placement decision service or script that will produce a deployment specification complete with mapping of the pipeline components to TEADAL Nodes. This service or script will have to rely both on the pipeline specification received from step 1 and on the infrastructure data, such as inventory information about clusters and their performance status.
- When deployment manifests are ready, GitLab action will continue the process and push the manifests to the GitLab repositories that back up their corresponding TEADAL Nodes.
- ArgoCD assigned to the TEADAL Nodes will notice configuration changes in the repository and will roll-up the corresponding changes to its cluster.

2.3.2 Updated TEADAL Data Pipelines

In this section we describe how TEADAL the Control Plane takes care of deploying the complex multi-component data pipelines as part of the stretched data lake. In the initial design described in D4.1 [4], this capability was based on two interdependent components, a Stretched Data Lake Compiler and a Stretched Pipeline Executor. The purpose of a Stretched Data Lake Compiler was to generate workload deployment decisions based on what data pipelines need to be installed and what are their runtime requirements. The purpose of a Stretched Pipeline Executor is to realize the workload deployment decisions generated by a Stretched Data Lake

Funded by the European Union

Compiler so that all the required components are deployed and available according to the requirements dictated by the Data Governance and by the Trust Management planes.

In the first project iteration, an IBM Research asset named Multi-cloud Computer Compiler or MCC-C, was used for a proof-of-concept (PoC) implementation of the TEADAL Stretched Data Lake Compiler. A custom version of MCC-C used for this initial PoC was contributed to the project and is now hosted in the TEADAL GitLab organization in a *stretched-data-lake-compiler*[6] repository. As the initial prototype for a Stretched Pipeline Executor, an open-source project Kubestellar created to manage workloads over multiple k8s clusters while itself being based on k8s, was used. This initial TEADAL data pipelines PoC has demonstrated the capability to orchestrate an FDP creation pipelines, for example the pipeline shown in Figure 11, as well as an sFDP creation pipelines, for example the pipeline shown in Figure 12, as part of the first period review.

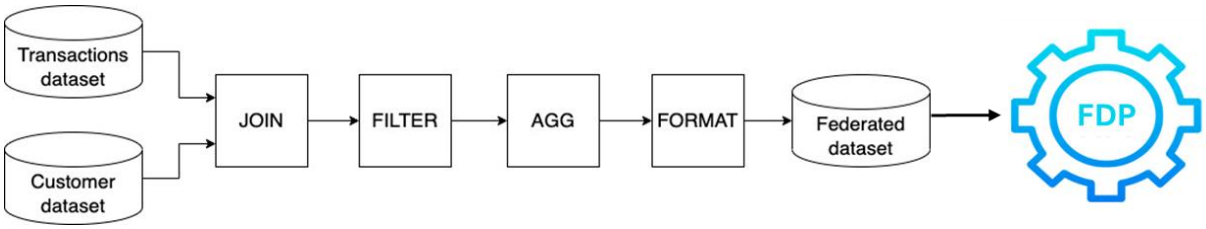

FIGURE 11: EXAMPLE OF A TEADAL PIPELINE FOR CREATING AN FDP

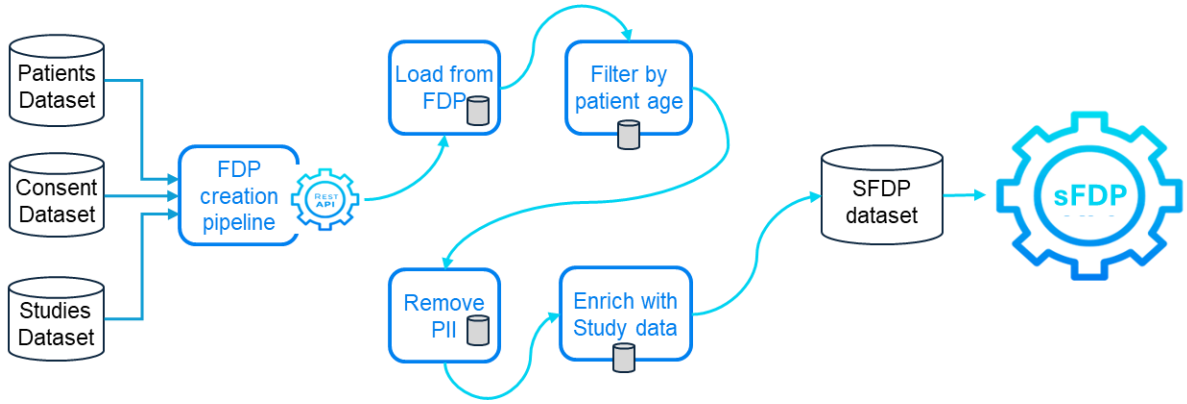

FIGURE 12: EXAMPLE OF A TEADAL PIPELINE FOR CREATING AN SFDP

The Stretched Data Lake Compiler optimizes workload placement based on data pipeline specifications that contain references to input datasets as well as the task deployment characteristics, both generic, such as input-output ratio, location constraints, hardware requirements, and TEADAL-specific, such as constraints computed by Data Governance and Trust Management planes. In addition to data pipeline specifications, the Stretched Data Lake Compiler needs infrastructure inventory information, annotated with properties such as compute and memory capacity, storage size, advanced hardware capabilities such as GPUs or TEEs available at infrastructure nodes, etc. In the first iteration prototype, mainly the generic annotations were used both for the data pipeline specification and for the infrastructure capabilities specification. Generic tools such as IBM MCC-C for the Stretched Data Lake

[6] https://gitlab.teadal.ubiwhere.com/teadal-tech/stretched-data-lake-compiler

Funded by the European Union

Compiler and Kubestellar for the Stretched Pipeline Executor were sufficient to realize the PoC. To summarize, the main outcome of the initial, first period, work, was to architect the stretched data lake control as a combination of two compoments, a Stretched Data Lake Compiler realized with the MCC-C and a Stretched Pipeline Executor planned to be realized with Kubestellar.

In the second project iteration, the architectural approach of having two separate components, the compiler for making placement decisions and executor, for actuating these decisions, was preserved. During the second iteration, several important advances took place in the project as a whole, both with respect to more detailed specification of the pilot requirements in WP2 and with respect to a progress achieved in the technical work packages. This resulted in the updated understanding of the TEADAL Control Plane. The principal separation of duties between the data pipeline optimization, prototyped as Stretched Data Lake Compiler, and the data pipeline execution, prototyped as Stretched Pipeline Executor, remains the same. At the same time, the functionalities and the realization mechanizms of these components were refined. In the current deliverable, we mostly focus on the updated executor functionality explained below.

2.3.3 Deploying Data Pipelines with GitOps

Data pipelines realize data-based computational transformations of the datasets, defined to create task-specific data products, for further consumption by the federation, based on the data governance as well on the security and trust rules and policies. From a technical standpoint, data pipelines can be seen as sequences of computational steps to be performed on specified data sources in a specified order. Although in the simplest cases, all the required transformation steps can be executed as a single monolithic application instance, a more modular approach is required for achieving an adequate balance between satisfying the optimization goals and standing in data governance and security constraints. Thus, data pipelines in TEADAL are multi-component applications with different components executed on different infrastructure pieces based on optimization and governance decisions.

To enable uniform deployment over different infrastructures, such as cloud IaaS, on-prem data centers and edge facilities, TEADAL Node architecture follows cloud-native principles and is realized using Kubernetes (k8) technology. Hence, each TEADAL Node is a k8s cluster extended with several layers of TEADAL services: 1) the Infrastructure Mesh services responsible for interconnecting individual TEADAL Nodes into a uniform infrastructure; 2) the TEADAL Core services responsible for enforcing data governance and trust policies, as well as for control plane and pipeline creation functionalities; and 3) the TEADAL Product services, such as REST endpoints of the federated and the shared data products that are available for consumption by the end users. According to the k8s and the service mesh architectures, all the software running on each specific TEADAL Node is controlled natively by the local k8s control plane according to a declarative definition of the k8s objects stored as cluster configuration in YAML files. TEADAL infrastructure is managed as code through the ArgoCD based GitOps processes developed as part of WP6. According to these processes, each TEADAL Node is backed up by its own git repository and is governed by its own ArgoCD daemon that synchronizes the Node's deployment state with the state declared by configuration files in the git repo. During the second project period, this TEADAL GitOps automation approach, based on ArgoCD and described in D6.2[7], was validated across the project pilots and found capable of managing the infrastructure mesh, the core, and the product services such as FDPs and sFDPs. It was decided to leverage the success of the created GitOps processes and extend them to manage all the service deployments, with no need for an additional deployment facility. This allows the GitOps automation to cover the functionality of the Stretched Pipeline Executor, significantly simplifying the control plane, reducing the number of deployed services and, thus, the resource and the energy requirements of the TEADAL framework.

Funded by the European Union

To realize the GitOps based Stretched Pipeline Executor functionality, we have considered several approaches, including GitLab automation, multi-cluster and multi-repo ArgoCD practices, as well as terraform. After the initial evaluation, it was decided to continue the second phase experimentation and deploy sFDP pipelines with GitOps as proposed in 2.3.1.3. For this, we plan to create, in addition to a per-node git repository, a federation-level git repository for each pilot. This repository will contain mostly configuration of the pilot federation infrastructure and services, in cloud-native declarative form. Updates to this federation-level repository will trigger updates to the per-node repositories and, as a result, the nodes backed up by these repositories will be updated by their local ArgoCD daemons.

Funded by the European Union

3. ADVANCE TEADAL AUTOMATION

In this section we describe the novel AI-based techniques for automating the TEADAL data handling and data sharing processes. This automation is proposed to reduce the complexity of sharing the data, and to avoid, as much as possible, the need for manual steps, such as developing custom software components, creating the configuration files, browsing through metadata in the catalogue, or specifying the data products based on policies and rules formulated by humans. These capabilities are new research that is introduced in the second part of the project and will be demonstrated in a proof-of-concept capacity for the selected use cases as part of TEADAL pilots.

The research is targeting two directions:

- The first is to reduce the complexity for the data consumer by providing tools to help consuming the data in a simple way, without the need to use additional tools.
- The second is to simplify the development of the sFDP mechanism and to reduce the effort that data producers need to invest in implementing the data sharing mechanism.

The foundation behind the automation is the availability of OpenAPI specifications [12] which is a format standard for describing HTTP APIs such as the one used in TEADAL, and open-source LLMs with function-calling[13][25] capability which is the ability of an LLM to associate the content of a user request with a need to call an external tools, e.g., a function from a set of available functions or a remote service, and to identify the correct set of parameters to pass to the target function or service.

The technology at the core of the automation is implemented in the Generative AI integration library (GIN) library. It leverages the foundation described above and adds enabling technologies to realize a solution. The details are discussed in the next section.

3.1 GENERATIVE AI INTEGRATION (GIN) LIBRARY

The idea behind GIN library is to simplify the use of data and reduce the need for a priori knowledge when developing data access code or performing data science, analytics, business intelligence, and other data usage tasks. GIN library enables the user to avoid the need to study the technical specification of the available data APIs, as well as the need to possess programming skills necessary for using REST APIs.

GIN is a Python library that facilitates access to data and can be called either programmatically or by using a command line interface. The library internal components are shown in Figure 13 and the interfaces to those components are listed below along with explaining the functionality they implement:

1. An interface component to import, index and retrieve API.
2. Natural language to API call functionality including:
    a. A translator component, to translate a natural language query such as “list the shipments sent in October 2024”, “get shipments for customer with id 1234” into a machine-readable specification (JSON) that can be easily transformed into a function call or REST API call, also known as tool-calling.

    b. A connector generator component, to generate the specification YAML (also called data connector specification) describing the REST query to perform and how to handle the output data.

Funded by the European Union

3. An interpreter component, to interpret the specification created by the generator component, perform a REST call to the specified API endpoint, transform the API response data to a request output format, and returns the output to the caller.

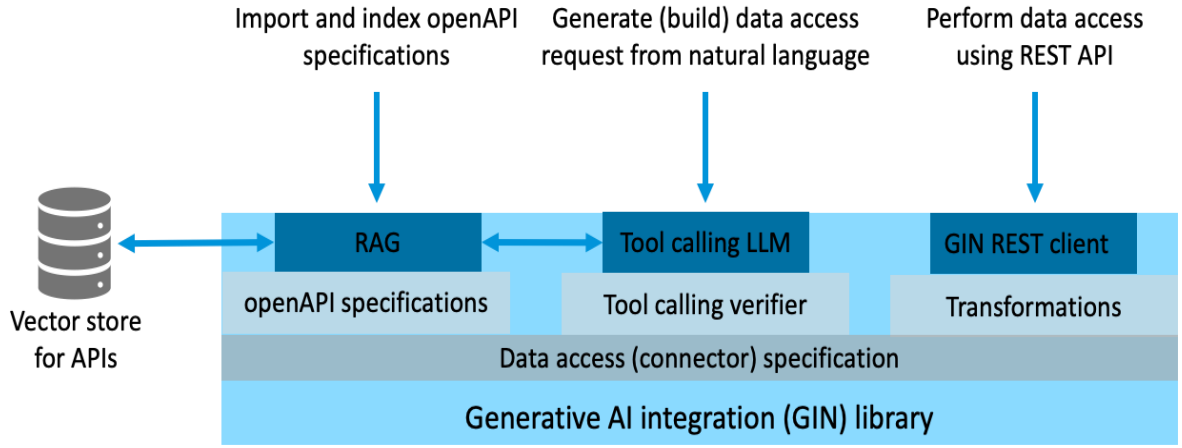

FIGURE 13: GIN LIBRARY COMPONENTS

GIN uses an LLM to translate an input query in a natural language to a function call or a REST API. The LLM is trained to select a best fit function from a given list of functions and to assign values to the function arguments from the input query. The LLM generates a JSON description for the function name to call and value assignments to the function arguments. To assemble the list of candidate functions GIN uses a pattern called Retrieval-Augmented Generation (RAG) to identify the most relevant functions or REST APIs that are to the input query.

GIN builds a data access (connector) specification YAML from the LLM output. The YAML describes the details of performing a REST API request and details on how to read and parse the response and how to transform the response content into the desired output.

3.1.1 Using Retrieval-Augmented Generation for OpenAPI specifications

GIN uses the IBM granite-20b-functioncalling LLM [14][27] to identify the function to call and to assign values to the function arguments. Such a function-calling LLM expects to receive an input context that includes the input query and a list of functions to choose from. The LLM generates an output that specifies what is the best fit function from the list that matches the input query.

GIN is built to support choosing from a large set of functions and APIs. However, the LLM input content is limited and needs to be as concise as possible and cannot include the full set of possible functions that are available. For this purpose, we identify the set of relevant APIs and only provide those to the LLM. This is known as Retrieval-Augmented Generation (RAG). In GIN we apply the RAG pattern by retrieving the most relevant function calls from a Chroma vector store [26].

Retrieval from a vector store is based on converting text into an embedding vector which is a list of floating-point numbers. The embedding vector is generated using sentence transformers which generates the embedding vector to represent the content of the input text. The distance between two vectors measures their relatedness. Small distances suggest a high level of relatedness and large distances suggest low level of relatedness.

GIN applies the RAG pattern by performing the following flow:

Funded by the European Union

4. Load function calls / APIs into the vector store. GIN creates a text document that includes the function name, description, the function argument and their description, type and additional information that can help with matching an input query. This document is converted to an embedding vector and stored in the vector store.
5. Convert the input query is converted to an embedding vector and used to search the vector store. The closest (small distance) set of vectors are fetched from the vector store including their metadata (i.e., the text document describing the function call).
6. Generate a list of function calls in the required format (described in the LLM documentation) from the metadata retrieved by the search.
7. Call the function calling LLM with the input context to get back the candidate function to call.

3.1.2 Function Calling LLM

The function calling LLM returns a JSON description of the function to call and the value assignment to arguments. However, LLMs, by their probabilistic nature and training, are prone to various types of issues when returning results, all the way from incorrectly selecting the function to introducing “hallucination” (i.e., combining or mixing unrelated data together).

For this purpose, GIN adds a validation layer on top of the function calling LLM. The validation layer adds checks for the LLM output. In this version, we introduce a layer of static checks on the LLM output. The checks stem from the OpenAPI specification and from the Tool Description, verifying the following:

1. The function name - the function selected by the LLM was included in the list of candidates.
2. The arguments – the arguments listed by the LLM are listed in the function call document.
3. The types – the values assigned by the LLM to arguments are correct according to the function call document specification.
4. The required arguments – all the required arguments according to the function calling document are included in the LLM output.

These sets of checks enable GIN to increase the confidence in the output of the LLM.

3.1.3 Data Transformations

The run-time engine that performs the REST API can transform the API response data and shape it into a given format before producing the output to the user.

The export section in the connector specification defines for each output field (field after transformation) the needed transformation functions to run on the API response fields to get this output field. An example of how to describe transformations is available later in Figure 15. For each output field that is provided, the functions in the list are executed in order. The name of the function is taken from “function” and the function arguments (i.e., argument name and value) are populated from the “params” section.

3.1.4 Using the GIN Library

Table 2 shows the methods expose by the GIN library, to be used either by the data consumer or by the sFDP builder.

TABLE 2 : GIN LIBRARY METHODS

Funded by the European Union

Method Name	Method Decription
`add_specs`	Import into the search database (a vector store database) APIs described in an OpenAPI specification file to be available for generating data access requests.
`apply_tool_calling`	Generate a function call description from an input query and a given list of function call candidates.
`build_connector`	Builds data access connector specification from natural language (in English) input query using the RAG pattern. Selects candidate functions based on the input query and generates a specification that describes how to access the data source and what arguments (and values) are needed to retrieve the requested data.
`run`	Performs a data access specification request using a REST engine that connects to a REST endpoint and retrieves the requested data.

Later in this section, we describe a set of capabilities designed to assist data users such as data scientists, data analysts, auditors, etc. The user persona depends on domain specific use cases of the TEADAL pilots. For example, when a data scientist needs certain data for a specific research, report, or model, the TEADAL framework aims to provide convenient interfaces to discover the suitable source FDPs, to specify how the data, served by the source FDPs, needs to be transformed and turned in an sFDP, and, finally, to access the data served by the sFDP prepared specifically to be used for a data science task at hand.

3.1.5 GIN Data Access (Connector) Request description

GIN uses a YAML specification to describe a data access request. The `build_connector` method builds the data access request, also called a connector by GIN, which describes how to build a REST API call to the request data endpoint.

The specification, example is given in Figure 14, includes the following information:

1. Metadata (some is omitted from the figure) such as the input query, headers, versions, etc.
2. The server addresses.
3. Technical specification for the API(s) to call, including the API endpoint, the method type (get, put, post), the arguments: for each argument it includes the name, the location (URL parameter, header, data), the type and value assignment.
4. The output data that should be returned by the call, including the transformation that needs to be performed.

Funded by the European Union

```
metadata:
         inputPrompt: get shipment of customer with id 1113
servers:
  - url: http://localhost:8003/fdp-czech-plant/
spec:
  apiCalls:
    .getShipmentsByCustomer:
      arguments:
      - argLocation: parameter
        name: id
        type: string
        value: '1113'
      endpoint: /shipments/customer/{id}
                              method: get
      type: url
  output:
    data:
      .getShipmentsByCustomer:
        api: .getShipmentsByCustomer
```

FIGURE 14: EXAMPLE DATA REQUEST (CONNECTOR) SPECIFICATION

In Figure 15 we show an example how to specify transformation in the YAML. It contains a "Shipment" entry which is the output schema after transformations, and it has two fields, "year" and "month", for each field there is exactly one transformation function with the needed parameters (the parameters are taken from the API output schema in the spec section).

```
"exports":
{
  "Shipment": {
    "dataframe": ".",
    "fields": {
      "year": [
        {
          "function": "map_field",
          "description": "map fields or change names from source to
target.",
          "params": {
            "source": "year",
            "target": "year"
          }
        }
      ],
      "month": [
        {
          "function": "filter_by_quarter",
          "description": "Filters a DataFrame to return rows where the
month falls within the specified quarter.",
          "params": {
            "month_col": "month",
            "quarter": 4
          }
        }
      ]
    }
  }
}
```

FIGURE 15: EXAMPLE TRANSFORM SECTION IN THE SPECIFICATION

Funded by the European Union

The GIN `run` method receives as input the data request specification YAML and performs the REST API call described in the specification and applies the transformations defined in the exports section to the API response data.

3.2 ASSISTING THE DATA CONSUMER

In this section, we describe a set of capabilities designed to assist data users such as data scientists, data analysts, auditors, etc. The user persona depends on domain specific use cases of the TEADAL pilots. For example, when a data scientist needs certain data for a specific research, report, or model, the TEADAL framework aims to provide convenient human-friendly interfaces and processes to find, select, and obtain datasets according to a natural language description instead of requiring data scientists to browse through huge amount of metadata that can involve lots of guesswork, frustration, and waste of time.

Two key complexities in the data user workflow toward working with data is (1) finding the data source (API) that would provide the data, and (2) the technical skill to access the data, that is writing code that would bring the data in the right form for processing. Leveraging GIN, a data consumer is able to find and obtain data without the need for deep skills at finding data, or at developing code to access data. In Figure 16 we show how GIN is used to assist the data consumer.

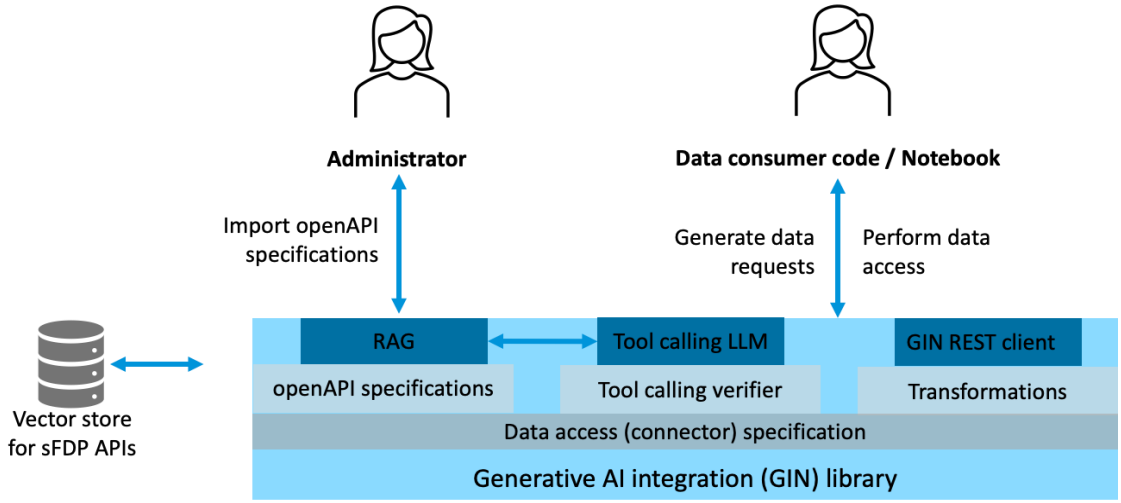

FIGURE 16: USING GIN TO ASSIST THE DATA CONSUMER

The `build_connector` method in GIN maps between a user query, phrased in a natural language (English), and data that is shared in TEADAL (through the use of TEADAL sFDP). Based on the content of the user query, the method finds an API call that is a best fit for the information provided in the query. More so, it also generates an assignment to any arguments/parameters that needs to be included in the API call to return the request information that was specified in the query.

The output of the `build_connector` method is a specification (in JSON) that describes how to call the API, and what information is retrieved from the API. See details in Section 3.1.4.

The `run` method in GIN takes the above specification, that describes how to instantiate a user query and executes the API call to retrieve the data based on the specification. It returns either a JSON or a Pandas dataframe that includes the retrieved data. It also has the option to produce a JSON file that can be further processed.

Funded by the European Union

The `add_specs` method imports a set of APIs described in an OpenAPI specification file into a vector store, which is used for searching and matching a natural language query text with and API.

3.2.1 Consuming data from sFDP using GIN

Here we show an example of how one can use the sFDP OpenAPI specification with GIN to simplify the access to TEADAL sFDP. The example is composed of two Python notebooks, one that loads descriptions of APIs from the provided OpenAPI specifications into a vector store, and the second that generates and performs an API call based on a user query. The Python notebook shows how one can consume data from TEADAL sFDP without the need for a deep knowledge of the sFDP APIs.

The notebooks are part of the repository that holds data consumer tools. Table 3 depicts the repository location and the available notebooks.

TABLE 3 : SAMPLE NOTEBOOKS FOR DATA CONSUMERS

Repository	**https://gitlab.teadal.ubiwhere.com/teadal-tech/sfdp_consumer_tools**	
Notebooks	load_vs_sample_notebook.ipynb	Load OpenAPI specification into the vector store
	data_access_sample_notebook.ipynb	Generate and execute the data access request

The data access notebook in Figure 17 shows how to access the sFDP that holds shipments data (based on use case pilot 4 – industry 4.0). The notebook details how to create a data access request by providing an English query asking for shipments for a given customer that retrieves the data and returns it as a JSON document. Note that by using the library the data user does not need to specify exact arguments to the API, but rather the library creates the mapping between the text the data user provides and the API parameters that needs to be passed to execute the query. The output JSON document can be used for further analysis.

```
from gin.gen import config

from gin.gen.agents import build_connector
from gin.gen.executor import exec
from gin.common.logging import Logging
                                        …
query = "get shipments for a customer with id 1113"
…
build_connector.generate(config_file,
                         None,
                         connector_file,
                         query)

out = exec.run_from_spec_file(connector_file)

pprint.pprint(out)
```

FIGURE 17: CODE EXTRACT FOR GENERATING AND RUNNING A DATA REQUEST

Prior to running the generating and performing data requests we need to have the APIs included into a vector store to enable searching and matching with the English queries. This

Funded by the European Union

action needs to be run once as the vector store that keeps the APIs is persistent and can be used for any additional runs. Figure 18 shows a code extract that describes how to load APIs for two of TEADAL uses, the mobility use-case and the industry 4.0 use-case.

```
from gin.gen import config
from gin.gen.retrieval import vectorstore_loader

specs_files = ["./openapi-specs/fdp-amts-gtfs-static.yaml", "./openapi-
specs/fdp-czech-plant.yaml"]

vectorstore_loader.add_specs(conf, specs_files
```

FIGURE 18: CODE EXTRACT FOR LOADING APIS INTO THE VECTOR STORE

3.3 ASSISTING THE SFDP BUILDER

The set of capabilities described in this section is designed to assist the participants of TEADAL federation that need to create deployment artifacts required to serve the data as TEADAL sFDPs. In the initial project phases, it was assumed that the TEADAL federation fully depends on dedicated development teams assigned for this task. The developers can be part of the data producer, data consumer, or federation provider organization. In the current project iteration, we advance our thinking to support, in addition to allowing sFDP creation by traditional software teams, allowing automatic sFDP generation by generative AI using LLMs.

In this section, we introduce the Automatic sFDP generation (ASG) tool that enables a sFDP developer to create a REST API server without writing computer code, but rather with providing ASG specification that describe the role of the sFDP including transformations that needs to be performed in order to uphold the contract between the data producer and data consumer.

3.3.1 Generating the Open API Specification for sFDP

The ASG specification describes the function of the sFDP. It describes which REST endpoints needs to be available, the REST endpoint of the FDP that provides the data for this sFDP, the response schema that should be included in the API response and the description how to generate the sFDP response based on the data received from the FDP.

In Figure 19 we see an example of ASG specification. The specification states that the sFDP should have an endpoint `/stop_id/{stop_id}` and it should return a JSON text that includes two fields, a `stop_ID` field, which is an integer and is populated from *stop_id* in the source FDP, and a `stop_full_name` field which is a string and is populated by concatenating `stop_name` and `parent_station from` the source FDP.

The description entry for each field in the schema describes how to populate the data in the sFDP, the description is used as input query to call GIN tool calling method together with the available library of functions and the FDP response schema in order to select which transformation to use, and which arguments it needs to provide the response data.

Funded by the European Union

```
sfdp_endpoints:

  - stops_endpoint:
      fdp_path: /stops/stop_id/{stop_id}
      sfdp_path: /stop_id/{stop_id}
      schema:
          Stop:
              type: object
              properties:
                stop_ID:
                    type: integer
                    example: 101
                    description: "map stop_id"
                stop_full_name:
                    type: string
                    example: Martiri LibertĂ SaittaMarshall
                    description: "Concatenated stop_name with
parent_station for this stop"
```

FIGURE 19: SAMPLE ASG SPECIFICATION BASED ON THE MOBILITY USE-CASE

3.3.2 Generating the sFDP API Endpoint

The ASG tool generates an sFDP server which is implemented as a FastAPI application from the ASG specification and the OpenAPI specification of the source FDP. The tool also includes a set of predefined transformation functions (tools) that is available to be used in the sFDP server.

The ASG tool can be used when a new contract is added to the catalogue, for example, by the sFDP developer who invokes the tool with two inputs:

1. The ASG specification file which defines the sFDP endpoints based on the contract. For each endpoint, the developer provides the FDP endpoints used and the transformations of the data in natural language.
2. The OpenAPI specification of the source FDP.

The ASG tool creates Python code for a FastAPI application, implementing the endpoints defined in the ASG specification with the needed transformations on the data retrieved from the FDP.

Alternatively, the process of generating the FastAPI application can be invoked automatically upon creation of a new contract or on other triggers, such as modification of the existing contract or of the source FDP API.

The ASG tool is a Python based tool that uses the following components:

1. FastAPI framework to implement a REST server.
2. Jinja2 templates to automatically generate the FastAPI server application code.
3. GIN tool calling to infer needed transformations and the sFDP response schema and data.
4. GIN Execution engine to retrieve the data from the FDP and transform the data before sending to the consumer.

In Figure 20 we describe the flow of creating a sFDP (depicted as flow 1) and the execution time of the FastAPI application (depicted as flow 2).

Funded by the European Union

ASG creates the sFDP FastAPI application (flow 1) as follows:

4. For each endpoint in the ASG specification:
    a. For each property defined in the schema
        i. Run `create_spec_section()` to parse the FDP OpenAPI specification and create a data access request specification section to retrieve data from the FDP.
        ii. Run `add_export_section()` to use the GIN tool calling to translate the field description written in natural language (English) to one of the transformations available in ASG, and generate the export section in the GIN data request specification which describes the transformation to run.

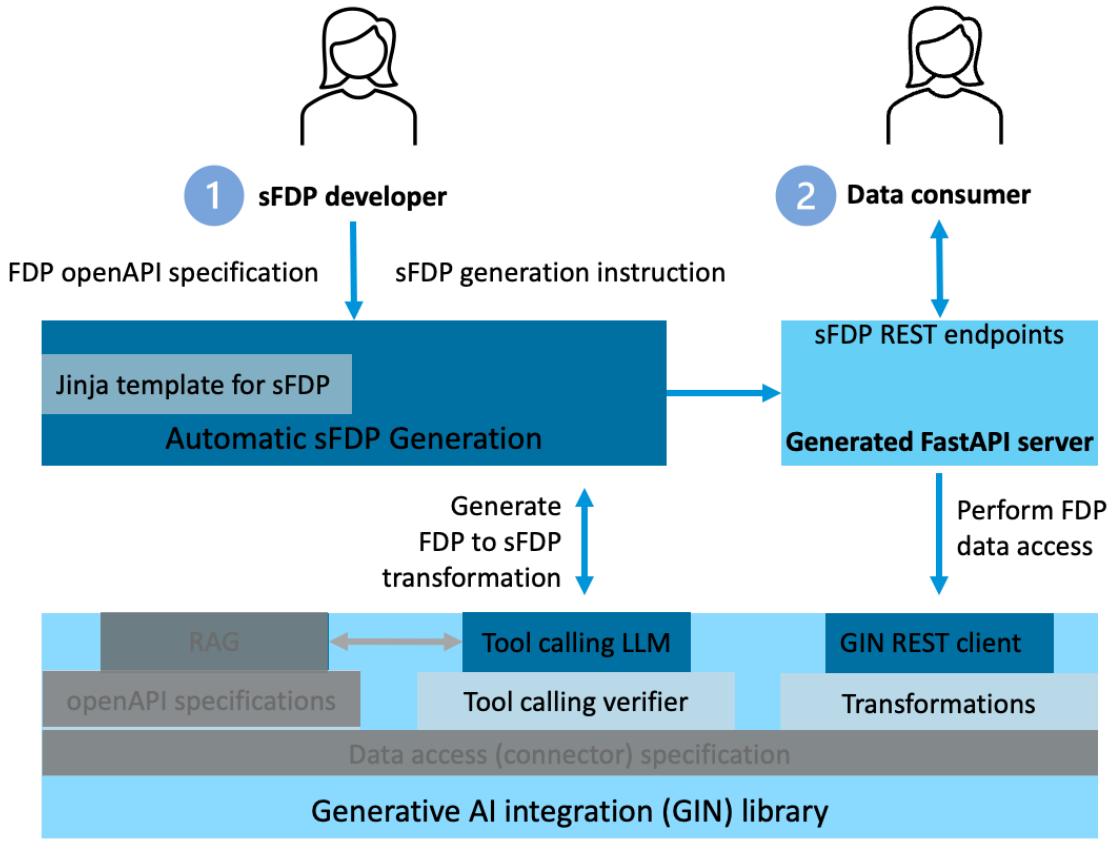

FIGURE 20: THE AUTOMATIC SFDP GENERATION TOOL

When the FastAPI application endpoint is called (flow 2), it uses the GIN execution engine using the GIN method `exec.run_from_spec_string()` to execute the specification that was generated by the ASG tool in order to retrieve the data from the FDP, transform it, and return in to the sFDP data consumer.

TABLE 4 : ASG REPOSITORY AND MODULE

Repository	**https://gitlab.teadal.ubiwhere.com/teadal-tech/asg_generation_code**	
ASG Module	fast_api_from_spec/generate_app_from_spec.py	Generate sFDP server

Funded by the European Union

An example with details how to run ASG to generate the sFDP server are detailed in Figure 21.

```
python generate_app_from_spec.py -spec .\openapi-specs\fdp-czech-
plant.yaml -i .\transform\shipments-instructions.yaml -fdp_server
http://fdp-server-address/
```

FIGURE 21: PYTHON CODE FOR RUNNING AN ASG TOOL

3.3.3 Generating the sFDP Transformation Pipeline

The ASG tool supports a library of transformations that are provided as Python functions. The library is created in advance with generic data transformations and can be easily expanded by adding additional functions into the library to support use-case specific transformations.

Funded by the European Union

4. ADVANCE CONTROL PLANE FEATURES AND SERVICES

In this section we summarize the control plane related features, considerations, and services brought up and developed as part of the second project period or updated since the initial design created in the first reporting period of the project. The topics discussed here are broadly subdivided into topics related to security, privacy, and compliance in Subsection 4.1 and topics related to performance and efficiency covered in Subsection 4.2.

4.1 SECURITY, PRIVACY AND COMPLIANCE

Security, privacy and compliance are features of the utmost importance for a data sharing platform such as TEADAL. Even inside a single organization, data collection, storage, and processing require to conform to policies and regulations related to ensuring the data is treated according to organizational and government rules and distributed only to authorised people and destination. To share data between multiple organizations even more care is required and there is often an additional complexity of bridging between different technologies and tools used by different organizations. In TEADAL, these aspects are researched as part of WP5. Here we describe how security, privacy and compliance assets conceived in WP5 are integrated into the TEADAL control plane to facilitate cross organizational and cross-location data sharing.

4.1.1 Authorization

This subsection describes the authorization mechanisms between the data consumer and the data product as it currently stands. In the first project period, the focus was mostly on FDP creation and access described in detail in D4.1[4].

During the second reporting period, the approach was generalized to suit both the FDP and the SFDP data products. The diagram in Figure 22 illustrates the conceptual security components and actors as well as their relationships involved in accessing TEADAL Data Products, both FDPs and sFDPs. Blue lines show data product creation and orange lines show data product access flows. The flows presented in Figure 22 are:

1. Data consumer makes a request to a data product, either FDP or sFDP.
2. The request is intercepted by an in-network proxy implemented as an Istio sidecar installed in TEADAL Nodes.
3. In-network interception proxy forwards request details to the policy enforcement point.
4. The policy enforcement point invokes the relevant policy decision service.
5. The policy decision service looks up the actual policies related to the request from the policy store. Note here that dynamic changes to policies will be adequately supported as part of this design.
6. Based on policy look up, the policy decision service decides whether the request is legitimate or not and informs the policy decision point on this decision.
7. The policy enforcement service commands the in-network interception proxy to forward the allowed requests to the target data product service and to halt the disallowed requests. In both cases, the decision is recorded to the audit log to support compliance requirements and to the network log to help debugging.

The interception proxy forwards the allowed requests and then sends the reply to the data consumer. For denied requests, the proxy can generate an explanatory reject message or can just halt the transaction, depending on how the system is set up.

Funded by the European Union

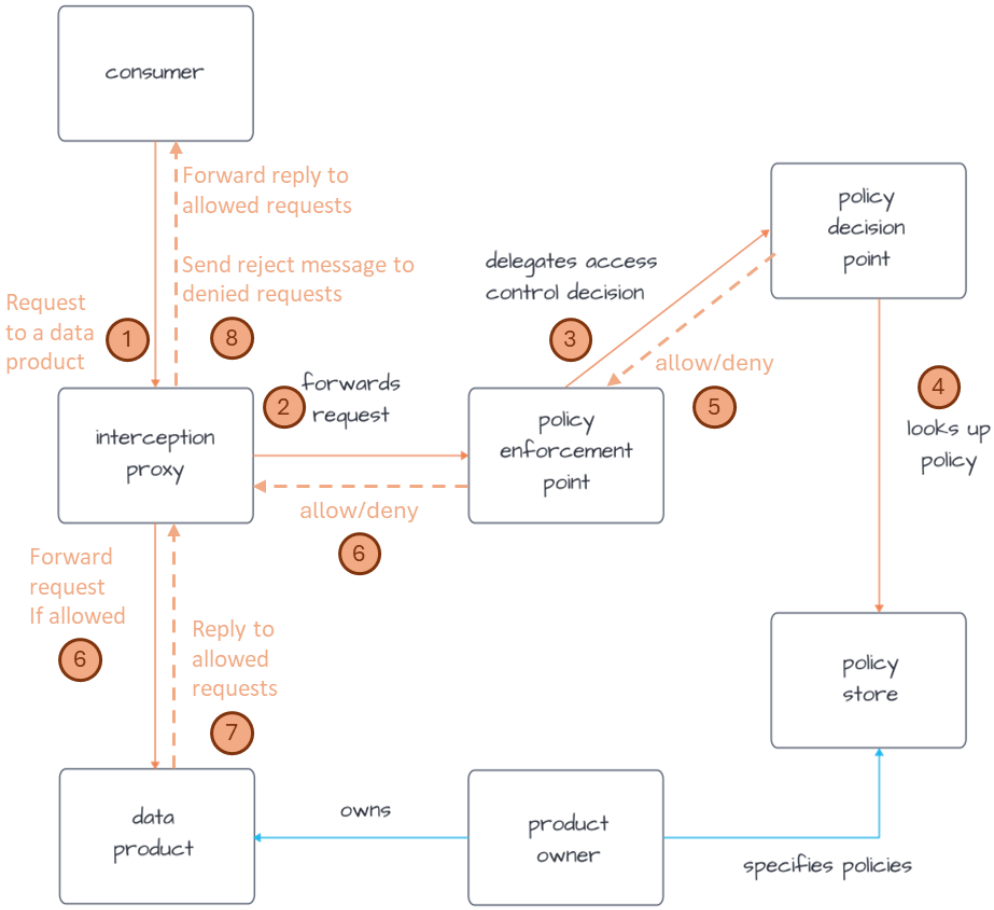

FIGURE 22: TEADAL DATA PRODUCT CREATION AND ACCESS FLOWS

The components currently implemented are:

Istio: Message interception facility is installed and configured in all the TEADAL Nodes. Istio's Envoy proxies intercept each consumer request and data product service response.
External Authorization Filter (Envoy): Envoy filter executed before forwarding consumer requests to a data product service. It acts as a policy enforcement point, connecting to an external policy decision point to determine whether to allow or deny data product service requests.
Open Policy Agent (OPA): A policy enforcement language, Rego, and versatile runtime for evaluating Rego policies both interactively and in server mode as well as testing Rego code and assembling it into binary, digitally signed "bundles" which can be downloaded and evaluated by the OPA server.
OPA Envoy Plug-in: Policy decision point. It embeds the OPA server runtime and interfaces with the External Authorization Filter. It downloads Rego policy bundles from an HTTP policy store.
Policy Store: Nginx application to create, sign and make available Rego policy bundles.
RBAC framework: Machinery to implement role-based access control for RESTful services, while still leaving policy writers the freedom to extend the base framework with service-specific functionality. It can be used with any Identity Management service compliant with the OpenID Connect (OIDC).
Keycloak: OIDC-compliant Identity Management service. Consumer services act on behalf of users who have proved their identity through Keycloak.

Funded by the European Union

At the time of writing, the message interception, access control delegation, OPA decision point and RBAC framework are in a functioning state.

The Nginx policy store has been partially implemented, where OPA can refer to it as a source for policies, and dynamically pull Policy Bundles from it. If the bundles are updated in the store, OPA will automatically pull them anew. If the bundles are signed, the signature can be verified by OPA as well, provided it is properly configured.

All policies are implemented using OPA's Rego language, with the RBAC (Role-Based Access Control) framework serving as the foundation for them. The framework is written to provide the base functionality to check requests towards RESTful APIs, verifying the JSON Web Tokens (JWTs) issued by Keycloak. In the previous iteration (D4.1 [4]), policy evaluations considered only roles, which were retrieved from Keycloak. Now, however, groups are also referenced and utilized for authentication, allowing requests to be approved based on both the roles associated with a user's identity and the groups they belong to in Keycloak. OIDC claims are now also supported in JWT evaluations.

In D4.1 [4] the possibility of using other policy evaluation solutions was discussed, such as Datalog. Ultimately, given the challenges of translating Datalog into usable access policy, and general inadequacy of Datalog for the purpose of the framework, OPA and Rego were ultimately selected for the authorisation solution presented here.

What is currently missing is a way to create Policy Bundles in a more structured and organized manner. A solution for this is being worked on, that involves providing an interface for manually creating policies for OPA, bundle them, and store them in the Policy Store, ready to be made available to OPA for evaluation. The plan is to have it functional and ready for a final project reporting for D4.3.

4.1.2 Orchestrating Privacy-Preserving Data Pipelines

To fully materialize the concept of Privacy-Preserving Data Pipelines introduced in D5.2 [6], we needed to make architectural decisions regarding workflow engines for executing data processing pipelines. After evaluating both Argo Workflows [40] and Kubeflow [41], we decided to proceed with Argo Workflows due to its flexibility and ease of integration with our existing infrastructure. Argo Workflows is an open-source container-native workflow engine designed for orchestrating parallel jobs on Kubernetes. It allows users to define workflows using simple, declarative YAML specifications. Integrating with Kubernetes, Argo Workflows can schedule and manage complex graph-like workflows, making it a valid choice for specifying and orchestrating data pipelines that require reliable, scalable, and programmable execution.

Argo Workflows can interplay with confidential computing mechanisms, for instance, provided by TEEs. As discussed in D5.2 [6], we evaluated different technologies for provisioning TEEs in Kubernetes, including KubeVirt[7] and Kata Containers[8] with Confidential Containers[9]. We decided to proceed with the Kata Containers and Confidential Containers approach due to its maturity and easier integration with the TEADAL architecture. Kata Containers provide lightweight virtual machines that are perceived and function like containers but offer the

[7] https://github.com/kubevirt
[8] https://github.com/kata-containers
[9] https://github.com/confidential-containers

Funded by the European Union

workload isolation and security advantages of regular VMs. When combined with Confidential Containers, they enable the execution of containerized workloads within protected VMs, leveraging various TEE hardware platforms. To use Kata Containers for TEE-based tasks, one can specify a runtime class as follows, part of an Argo Workflow pipeline specification in YAML like the one presented in Figure 23

```
- name: confidential-task
  template: tee-template
  ...
  - name: confidential-task
    podSpecPatch: '{"runtimeClassName": "kata-qemu-tdx"}'
```

FIGURE 23: ARGO WORKFLOW SPECIFICATION YAML

Here, `kata-qemu-tdx` is a runtime class that uses Kata Containers with the Intel TDX (Trust Domain Extensions) for hardware-based isolation, to ensure that this specific task runs in a secure enclave. KubeVirt was also considered, however, it operates differently from Kata Containers. KubeVirt extends Kubernetes by adding virtual machine resource types through Custom Resource Definitions (CRDs), allowing VMs to run alongside containers in a Kubernetes cluster. Kata Containers provide lightweight, isolated execution environment requiring minimal code changes, while KubeVirt extends the Kubernetes API for full VM management. Given our need for a simple hardware-based isolation of a single task within a pipeline, Kata emerged as the more suitable solution. KubeVirt's broader scope and deeper integration would have been unnecessarily complex for TEADAL's specific use cases.

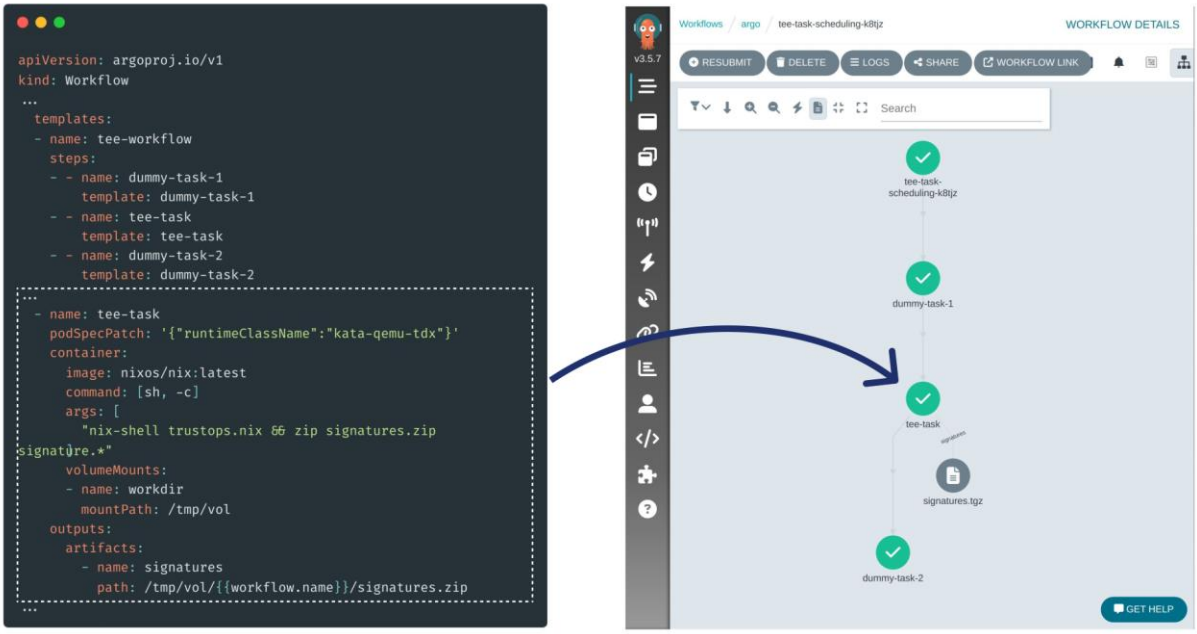

FIGURE 24: ARGO WORKFLOW PIPELINE EXAMPLE

Using the Kata Containers runtime class in Argo Workflows allows us to programmatically specify what specific and sensitive tasks in broader data pipelines should be executed within a TEE, maintaining its confidentiality throughout the processing pipeline, even when operating in potentially untrusted environments. Figure 24 presents an example of an Argo Workflow definition for a pipeline containing one TEE-based task (on the left), along with the visualization of this pipeline execution (on the right). D5.2 [6] further details the architectural components of Kata and Confidential Containers, running a VM-based TEE task setup. Building upon this

Funded by the European Union

implementation of Privacy-Preserving Data Pipelines, careful consideration must be given to the flow of data between regular and TEE-protected tasks. When data transitions into a TEE-protected task, it needs to be encrypted and securely transferred into the TEE, where it should remain encrypted in memory, only decrypted for processing. Results must be encrypted before leaving the TEE, ensuring data protection throughout the pipeline. The isolation provided by Kata Containers should enforce security boundaries between regular and TEE tasks, with each TEE task running in its own lightweight VM to prevent unauthorized access. To verify the integrity of TEE environments, a remote attestation process is necessary. When a TEE task initiates, it should perform a local attestation, generating evidence of its secure state. This evidence then needs to be verified by a trusted service within the pipeline or an external attestation service. Argo Workflows should integrate this attestation process, or its event sourcing, ensuring that sensitive data only flows into verified secure environments, thus maintaining the pipeline's trust model and ordered process execution.

As demonstrated in D5.2 [6] 's final section, this architecture can be applied to several TEADAL pilots, as well as other use cases, for example, continuous integration pipelines, part of TrustOps processes. In the former financial pilot use case prototype, for instance, sensitive data from Turkish citizens was demonstrated to be processed in TEEs hosted in the Netherlands, enabling cross-border data analysis while complying with data locality requirements. The TEE environment should undergo attestation by Turkish authorities before data transfer and processing occur within the secure enclave, allowing for secure aggregation and analysis of financial data across jurisdictions without compromising individual privacy or violating data protection regulations. In D3.2 [5], new possibilities for optimizing resource utilization and energy consumption without compromising data privacy are also detailed. To generalize and relate to the rest of this deliverable, this architecture is then relevant for processing FDPs and sFDPs within TEADAL. When creating or consuming sFDPs, which may involve sensitive data transformations agreed upon in data sharing contracts, this pipeline architecture can help ensure that these transformations occur within TEE-protected tasks. For instance, when an FDP is being transformed into an sFDP based on a specific data sharing agreement, a set of transformation logic can be encapsulated within a set of TEE tasks in a pipeline orchestrated by Argo Workflows. This ensures that the original FDP data remains confidential and that the agreed-upon transformations are performed in a verifiable, secure environment before the resulting sFDP is made available to the consumer. The attestation process becomes especially important in this context, as it provides assurance to both the FDP provider and the sFDP consumer that the agreed-upon transformations were executed correctly and securely.

The flexible nature of this pipeline architecture allows for optimized placement of FDP and sFDP processing tasks across the federated data lake infrastructure. Leveraging Kubernetes' capabilities in conjunction with TEE-enabled tasks, we can achieve a balance between data locality, performance, and security. For instance, certain preprocessing steps of an FDP might be performed closer to the data source for efficiency, while the sensitive transformations required to create an sFDP can be scheduled on nodes with TEE capabilities, regardless of their physical location. This approach enables TEADAL to maintain certain security guarantees for sensitive operations while still allowing for efficient use of distributed resources across the federation. Furthermore, this flexibility extends to scenarios where different parts of an FDP or sFDP pipeline have varying security requirements. Non-sensitive portions of the workflow can be scheduled on regular nodes, reserving the TEE-enabled resources for the most critical operations. This granular control over task placement and security levels, effectively handled by the workflow engine, allows for more efficient resource utilization across the federated data lake, potentially reducing energy consumption and operational costs without compromising on the security of sensitive data transformations. To achieve and further enhance this architecture, more sophisticated Kubernetes placement strategies, such as node affinity and taints, should be incorporated to optimize TEE task allocation. We will keep exploring and demonstrating Argo Workflows' advanced pipeline control flow mechanisms to create more

Funded by the European Union

complex and flexible privacy-preserving workflows. Additionally, to address the need for handling continuous data streams, investigation into Argo's daemon containers to support long-lived pipeline tasks will be necessary, as these are ideal for continuous streaming applications. These future enhancements, left for the last iteration of TEADAL, will purposefully aim to further improve the flexibility, efficiency, and security of Privacy-Preserving Data Pipelines, enabling TEADAL to handle an even broader range of data processing scenarios while maintaining strong privacy guarantees.

It is important to note that while TEEs currently represent the most promising and mature technology for implementing Privacy Preserving Data Pipelines in TEADAL, we are also actively developing prototypes leveraging other advanced privacy-enhancing technologies. Specifically, we are exploring the use of Secure Multi-Party Computation (MPC) and Zero-Knowledge Proofs (ZKPs), as planned in D5.2 [6]. An MPC demonstrator is also being developed and integrated with Argo Workflows, with a particular focus on the Healthcare pilot, where it can enable secure collaborative computations on sensitive medical data across multiple organizations. Meanwhile, ZKPs are being integrated into monitoring pipelines, as described in both D3.2 [5] and D5.2 [6], to provide verifiable evidence of monitoring data and compliance with policies, without revealing sensitive information. These prototypes aim to demonstrate the versatility of TEADAL's approach to privacy-preserving computation, showcasing how different technologies can be applied to address varying privacy requirements across different use cases and within the TEADAL infrastructure itself. These prototypes eventually complement our TEE-based solutions, to offer a toolkit for secure and privacy-preserving data processing in federated data lake environments.

4.2 PERFORMANCE AND EFFICIENCY

Achieving the cross-organizational data sharing goals while ensuring optimal use of resources to facilitate both the application performance and the operational efficiency, is one of the main TEADAL aspects. This aspect was discussed in detail in the previous deliverable of this work package where the basic methodology of multi-location resource optimization based on dynamically collected metadata was presented. In this second release, we discuss additional topics: first, subsection 4.2.1 presents our approach to improving efficiency by transforming raw data into more resource efficient formats; next, subsection 4.2.2 discusses the resource utilization and efficiency aspects related to employing AI techniques, and LLMs in particular, as part of the TEADAL platform.

4.2.1 Data Compression and Transformations for Efficiency

Dealing with large-volume FDPs and transforming them into sFDPs can cause multiple issues in terms of storage, processing capacity needed and, more crucially, energy consumed in order to perform these transformations. While reducing the volume of the data needed to create a valuable sFDP might not always be possible, especially when the sFPDs are used for data analytics, ML and AI, research has shown that transforming the raw collected data into new formats using techniques such as data compression [15], [16], or more efficient data representations such as those aimed at highly-sparse and homogeneous datasets [17][18], can help in reducing the storage and transmission volume of the data.

This in turn would reduce the amount of energy required to build and share the sFDP and would further help in speeding up the processing of the sFDP once it reaches the data consumer. As such, we have started researching relevant techniques such as data compression, data aggregation, data encoding and sparse representations of high-dimensional data. The most adequate ones, or a combination thereof, will be selected as a basis for the work to be developed in the remainder of WP4.

Funded by the European Union

The result of this development will be an optional processing module that sFDP builders can use in order to reduce the volume of their datasets, without affecting the value of their FDPs to the sFDP consumers. The module will include multiple techniques for data transformations that sFDP builders can choose to apply to their FDP, thus aiding them in improving their data sharing capabilities and consumption.

4.2.2 Efficiency Considerations related to use of AI

As was presented in Sections 2.1 and 3.1 above, novel AI tools technologies can be very useful for achieving the respective goals of timely and efficiently analysing the metadata to produce useful insights in the former case and of automating manual steps in creating and consuming the main TEADAL data pipelines in the latter. Still, applying most if not all the AI tools is considered costly in terms or resource and time consumption so that employing AI as part of a TEADAL platform can seem counterproductive to the clearly stated goals of achieving the best possible performance and efficiency. In this section, we address this concern and explain why the proposed use of AI is beneficial to the project.

4.2.2.1 Impact Related to AI-OPs

The AI-based techniques introduced in Section2.1 are necessary to analysing large amounts of performance data at runtime and producing actionable insights. The use of AI for operational data analytics and optimization has already proven itself and is not disputed. This is because most of the resource savvy ML computations, such as experimentation and model training are performed offline and do not contribute to the runtime overheads of the system that makes use of the resulting models for decision making and inference. After all, the AIOps systems are usually put in place for improving the overall system performance and must be designed in a way that their computational, memory, and energy overheads are smaller than the resulting performance benefit for the system they are employed in.

The question is whether adding a typically more expensive generative AI technologies such as LLMs is beneficial. This question is still under research and the project team is actively seeking for ways to leverage these novel capabilities without incurring additional overhead that will overweight the benefits. Final report of this research outcomes is due in D4.3. Here we can share the general principles that guide us in augmenting AIOps toolbox with generative AI technologies, that is: 1. separate the heavy-duty resource-hungry steps such as data preparation, model training, etc, from the runtime decision making (inference); and 2. Keep the models as small as possible to be relevant to use cases at hand.

4.2.2.2 Impact Related to Automatic sFDP Generation (ASG)

The AI-based techniques introduced in Section 3 reduce the complexity of sharing data, simplify the effort for the data consumers and accelerate the creation of sFDPs for data sharing. On the other hand, it introduces resource and energy costs by requiring additional heavy-duty computing, including the use of GPUs. The GIN library makes use of small-scale LLMs for generating vector embeddings (Slate 30M) and function-calling (Granite 20B) to enable running GIN without requiring the use of high-end hardware that is required by high-end multi-billion parameter LLMs.

In this section, we review the use of AI and its impact on day-to-day use. The GIN library is built to leverage AI when translating natural language into explicit computer-generated instructions, and after the translation is done, the translation (i.e., specification) can be used repeatedly. This separates the effort into offline design-time where the user builds a data request or sFDP code, and an online run-time where the user can execute multiple times an efficient machine code.

Funded by the European Union

4.2.2.3 Impact Related to Assisting the Data Consumer

The data consumer uses the generative AI based logic in GIN when developing a data access request, overcoming the need to understand the details of API specification and REST mechanisms. As soon as the data access request is ready, the generated specification can be used by the run-time engine in GIN and can be applied efficiently without the need to use GPUs at runtime.

While the generative AI process requires a higher use of resources, including use of GPUs, the run-time flow strictly requires only the use of (efficient) CPU resources for supporting the data consumer.

4.2.2.4 Impact Related to Assiting The sFDP Builder

The sFDP builder runs ASG to develop and build the FastAPI server code for the sFDP. Generative AI logic is responsible for translating the instructions, written in natural language, that represent the contract between the data producer and data consumer into a set of transformations that can be executed efficiently during run-time.

ASG creates FastAPI server code that does not require any AI logic, but rather uses the GIN REST API engine to connect to the FDP, retrieve the data, and transform it according to the instructions (Python methods) specified in the connector (i.e., data access request) specification.

The sFDP code is expected to be generated once (or a small number of times) for each shared data, and hence requires a higher use of resources, including use of GPUs, the sFDP server code itself will be used frequently, but it strictly requires only the use of (efficient) CPU resources for sharing data with the data consumer.

Funded by the European Union

5. TOWARDS THE NEXT ITERATION

In this document, we summarized the second iteration of the TEADAL data lake which is focused on automating the data handling and the data sharing processes defined by the TEADAL architecture and pilots (D2.3 [3]).

This document has described the next step refinement of TEADAL control plane architecture over what was described in D4.1 [4], the new research-driven capabilities for automating the manual software creation steps defined as part of the TEADAL data sharing (D2.1 [1]], and D2.2 [1][2]), and the advanced capabilities related to the second phase focus on optimizations related energy efficiency (D3.2 [5]) and trust (D5.2 [6]).

Going forward, capabilities described here will be validated as part of pilots' integration towards the Milestone 4 (the final TEADAL architecture) that will be delivered as part of the final project demonstrators as well as of the next deliverable, D4.3.

Funded by the European Union

REFERENCES

[1] TEADAL Consortium, D2.1 REQUIREMENTS OF THE PILOT CASES, 2023

[2] TEADAL Consortium , D2.2 PILOT CASES' INTERMEDIATE DESCRIPTION AND INITIAL ARCHITECTURE OF THE PLATFORM, 2023

[3] TEADAL Consortium, D2.3 PILOT CASES' FINAL DESCRIPTION AND INTERMEDIATE ARCHITECTURE OF THE PLATFORM, 2024

[4] TEADAL Consortium, D4.1 STRETCHED DATA LAKES FIRST RELEASE REPORT, 2024

[5] TEADAL Consortium, D3.2 Reducing energy footprint in federated stretched data lakes, 2024

[6] TEADAL Consortium, D5.2 TRUSTWORTHY DATA LAKES FEDERATION SECOND RELEASE REPORT, 2024

[7] TEADAL Consortium, D6.2 Integration Report, 2024

[8] Hochreiter, S., & Schmidhuber, J. Long short-term memory Neural computation, 1997. 9 (8): 1735–1780.

[9] Cho, K., van Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. arXiv. https://arxiv.org/abs/1406.1078

[10] Ekambaram, Vijay, et al. "TTMs: Fast Multi-level Tiny Time Mixers for Improved Zero-shot and Few-shot Forecasting of Multivariate Time Series." arXiv preprint arXiv:2401.03955 (2024).

[11] Add reference: A. Detti, L. Funari and L. Petrucci, "μBench: An Open-Source Factory of Benchmark Microservice Applications," in IEEE Transactions on Parallel and Distributed Systems, vol. 34, no. 3, pp. 968-980, 1 March 2023, doi: 10.1109/TPDS.2023.3236447.

[12] OpenAPI Specification v3.1.1. https://spec.openapis.org/oas/latest.html. October 2024.

[13] Function Calling - OpenAI API. https://platform.openai.com/docs/guides/function-calling. 2024.

[14] IBM Granite led open-source LLMs on API calling. https://research.ibm.com/blog/granite-function-calling. August 2024.

[15] Azar, J. et al. An energy efficient IoT data compression approach for edge machine learning. Future Generation Computer Systems. vol. 96, pp. 168–175 (2019). https://doi.org/10.1016/j.future.2019.02.005

[16] N. G. Larrakoetxea et al. Efficient Machine Learning on Edge Computing Through Data Compression Techniques. IEEE Access. vol. 11, pp.31676–31685 (2023). https://doi.org/10.1109/ACCESS.2023.3263391

[17] J. Chen, S. Yang, Z. Wang and H. Mao. Efficient Sparse Representation for Learning With High-Dimensional Data. IEEE Transactions on Neural Networks and Learning Systems. vol. 34, no. 8, pp. 4208–4222 (2023). https://doi.org/10.1109/TNNLS.2021.3119278

[18] A. Ruospo, E. Sanchez, M. Traiola, I. O'Connor, A. Bosio. Investigating data representation for efficient and reliable Convolutional Neural Networks. Microprocessors and Microsystems. vol. 86, 104318 (2021). https://doi.org/10.1016/j.micpro.2021.104318

Funded by the European Union

[19] Prometheus. Power your metrics and alerting with the leading open-source monitoring solution, https://prometheus.io/.

[20] Grafana Labs. Grafana: The open observability platform. https://grafana.com/

[21] Thanos - Highly available Prometheus setup with long term storage capabilities. https://thanos.io/

[22] OpenTSDB - A Distributed, Scalable Monitoring System. http://opentsdb.net/

[23] Prophet | Forecasting at scale. https://facebook.github.io/prophet/

[24] OpenAPI Initiative. https://www.openapis.org/

[25] What Are Tools Anyway? A Survey from the Language Model Perspective, Zhiruo Wang, Zhoujun Cheng, Hao Zhu,Daniel Fried and Graham Neubig, 2024. https://arxiv.org/abs/2403.15452

[26] Chroma - the open-source embedding database. https://github.com/chroma-core/chroma

[27] Open-source granite-20b-functioncalling function calling LLM. https://huggingface.co/ibm-granite/granite-20b-functioncalling

[28] Kubernetes Cluster Federation. https://github.com/kubernetes-retired/kubefed

[29] Day 24: Kubernetes Multi-Cluster Management | by Vinoth Subbiah | Medium. https://medium.com/@vinoji2005/day-24-kubernetes-multi-cluster-management-7e53dfe465dd

[30] Kubernetes Cluster Federation: A Practical Guide. Overcast blog. https://overcast.blog/kubernetes-cluster-federation-a-practical-guide-f730af724762

[31] Open Cluster Management. https://open-cluster-management.io/

[32] Open, Multi-Cloud, Multi-Cluster Kubernetes Orchestration. Karmada. https://karmada.io/

[33] Multi-Cluster Kubernetes. Simplified. | Admiralty. https://admiralty.io/

[34] KubeAdmiral. https://kubeadmiral.io/

[35] KubeAdmiral: next-generation multi-cluster orchestration engine based on Kubernetes. CNCF. https://www.cncf.io/blog/2023/11/24/kubeadmiral-next-generation-multi-cluster-orchestration-engine-based-on-kubernetes/

[36] Landing - KubeStellar. https://docs.kubestellar.io/

[37] KubeStellar Explainer - New Architecture - Same Functionality. https://www.youtube.com/watch?v=M7yh1Wx-J2A

[38] The Cluster API Book. https://cluster-api.sigs.k8s.io/

[39] Workload bootstrap using GitOps. https://cluster-api.sigs.k8s.io/tasks/workload-bootstrap-gitops.

[40] Argo Workflows. https://argoproj.github.io/workflows/

[41] Kubeflow. https://www.kubeflow.org/

Funded by the European Union