TEADAL

D4.1 STRETCHED DATA LAKES

first release report

Revision: v.1.0

Work package	WP 4	
Task	Task 4.1, 4.2, 4.3	
Due date	31/01/2024 (extended)	
Submission date	11/01/2024	
Deliverable lead	IBM	
Version	1.0	
Authors	Ronen Kat (IBM) Josep Sampe (IBM) Bruno Wassermann (IBM) Andrea Falconi (MARTEL) Gabriele Cerfoglio (MARTEL) Giacomo Inches (MARTEL)	Sergio Sestili (ALMAVIVA) Antonio Retico (ALMAVIVA) Olango Temesgen Magule (ALMAVIVA) Fabio Previtali (ALMAVIVA) Alessio Carenini (CEFRIEL)
Reviewers	Pierluigi Plebani (POLIMI) Sebastian Werner (TUB)	
Abstract	Technical summary of the initial iteration of the stretched data lake, detailing the architecture capabilities and components of the first iteration.	
Keywords	Control plane, Data lake, multi-cloud, multi-cluster, data flows	

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	4/12/2023	First version for feedback	Ronen Kat (IBM) Josep Sampe (IBM) Bruno Wassermann (IBM) Andrea Falconi (MARTEL) Gabriele Cerfoglio (MARTEL) Giacomo Inches (MARTEL) Sergio Sestili (ALMAVIVA) Antonio Retico (ALMAVIVA) Olango Temesgen Magule (ALMAVIVA) Fabio Previtali (ALMAVIVA) Alessio Carenini (CEFRIEL)
V0.2	8/12/2023	Document ready for review	Ronen Kat (IBM)
V0.3	28/12/2023	Review revision 1 completed	Ronen Kat (IBM) Bruno Wassermann (IBM)
V0.4	11/01/2024	Review revision 2 completed	Josep Sampe (IBM) Ronen Kat (IBM) Antonio Retico (ALMAVIVA) Alessio Carenini (CEFRIEL)
V1.0	11/01/2024	Finalized formatting	Ronen Kat (IBM)

DISCLAIMER

Funded by the European Union

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.

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*	

EXECUTIVE SUMMARY

This document reports the results of the activities of the first 15 months for WP4, detailing the results of the first iteration of the TEADAL data lake. In general, the work package addresses the challenges of managing and controlling data processing in the stretched (multi-location) data lake while not adding complexity — but rather, simplifying how data is used and shared in the data lake.

Specifically, this document describes the core control plane concept and mechanisms we apply to enable the data lake to be realized on a cloud continuum, which is composed of multiple locations such as locations running at edge locations, private cloud, public cloud). All while TEADAL software runs oblivious to the details of running on a partitioned (i.e., multi-location) infrastructure. It enables the TEADAL data lake to address the key challenges such as: management of multiple locations, optimizing the use of resources across these locations and controlling data usage across multiple locations, and allowing TEADAL Federated Data Products, which are used to share data with external consumers, to be easily deployed and managed.

The TEADAL data lake is built on top of industry standard Kubernetes approach, which enables TEADAL to leverage the rich Kubernetes ecosystem of tools and solutions, while enabling the Kubernetes concepts to be expanded into multiple clouds in a seamless way for the data lake workloads. Namely we show how (1) we use a centralized control for multiple locations on the cloud continuum, and (2) leverage an industry standard API to implement the multi-location data lake.

Together with the centralized control plane the document describes the technical details of how Federated Data Products are deployed and accessed, and how access is controlled.

To complement the control plane, the introduction of TEADAL pipelines allows data to be processed in the multi-location data lake without the need to consciously take care of multiple locations. The TEADAL pipelines extends Kubeflow Pipelines from a single cluster/location solution into a multi-cluster/location solution allowing Kubeflow Pipelines to seamless overcome complexities of multiple locations. The TEADAL pipelines facilitate the creation of the shared Federated Data Product which is accessed by external consumers.

Furthermore, a multi-layer metadata collection and optimization mechanisms are introduced. At the platform level TEADAL monitors and collects usage and performance information in order to turn into insight metadata that helps to optimize TEADAL pipelines. Then, at the data level, TEADAL observes technical metadata that is associated with the data used in the pipelines to optimize and address data gravity and data friction constraints.

Funded by
the European Union

TABLE OF CONTENTS

Funded by the European Union

Funded by
the European Union

LIST OF FIGURES

Funded by
the European Union

LIST OF TABLES

Funded by
the European Union

ABBREVIATIONS

ADMS	Asset Description Metadata Schema
API	Application Programming Interface
BPMN	Business Process Model and Notation
CD	Continuous Delivery
CI	Continuous Integration
DCAT	Data Catalog Vocabulary
FDP	Federated Data Product
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IT	Information Technology
I/O	Input / Output
JWT	JSON Web Token
LLM	Large Language Models
OIDC	OpenID Connect
OPA	Open Policy Agent
RBAC	Role-based access control
RDF	Resource Description Framework
REST	REpresentational State Transfer
SDLC	Stretched Data Lake Compiler
SFDP	Shared Federated Data Product
SQL	Structured query language
TMC	Technical Metadata Catalog
UDF	User Defined Function
WAC	Web Access Control
YAML	Yet Another Markup Language

….

Funded by
the European Union

1. INTRODUCTION

Data is the new oil, and organizations refine and process data in order to optimize its value. Also, like oil, data is not concentrated in a single place. In large organizations, each department (or business unit) may store data relevant to the operations and processes in its purview in different data management systems. More so, such data management systems are implemented on a mix of on-premises and cloud resources. Some data, which can help achieve business goals and support process improvements, is owned by external organizations.

In order to bring together data and derive value from it, one typically runs data integration pipelines on one or more data sources and produces data products — just like oil, data requires refinements before delivering value. Data pipelines that support modern AI workflows and corresponding data products, such as training and fine-tuning of large language models or cross-border analyses of customer transactions, consume large amounts of data from a variety of sources. Data pipelines need to integrate data that is spread across on-premises data centers as well as those of multiple cloud service providers.

Organizations use data lake platforms as a key mechanism to process and share information within the organization. Here we use the term data lake casually and refer to the myriad of technologies and approaches that organizations use to collect, store, share and gain business (or other) insight from data. Today's data lake has emerged as the mechanism to store and organize vast amounts of data in an enterprise. The data lake offers support for storing, managing, and processing both structured and unstructured data and can serve as a repository of all kinds of data in the enterprise.

In TEADAL the boundaries of the data lake are expanded beyond one location to a cloud continuum that includes multiple locations and environments (i.e., edge, on-premises, and cloud) enabling access and processing of data without restriction of location while observing data governance restrictions and data sharing agreements.

As shown in Figure 1 our focus is a **stretched data lake** model that is applied across multiple locations and environments in the cloud continuum. Thus, data would be available for access or processing in all data lake locations, all while making sure that data governance which imposes constraints is addressed. In TEADAL we categories data governance aspects into gravity and friction aspects. Data gravity concerns stems from the location and size of the data, and data friction concerns originates from the fact that data is passing between location and organizations. Details on data governance are available in "Deliverable 3.1 – Gravity and Friction based data governance"".

As TEADAL is also a data lake federation we differentiate locations by the organization that provides the location. That is, the organization that owns the compute and storage resources associated with the location. Between such locations there are data friction concerns that stems from ownership of the data, associated responsibilities as data owner, costs for resources to the data, etc. This differentiation translates into restrictions on using the location for storage and processing of the data. For example, an organization *B* would like to consume data from the data lake, so he provides resources for the TEADAL data lake. These resources (i.e., storage and compute) are to be used (if needed) for consuming the data that organization B would like to consume.

Funded by the European Union

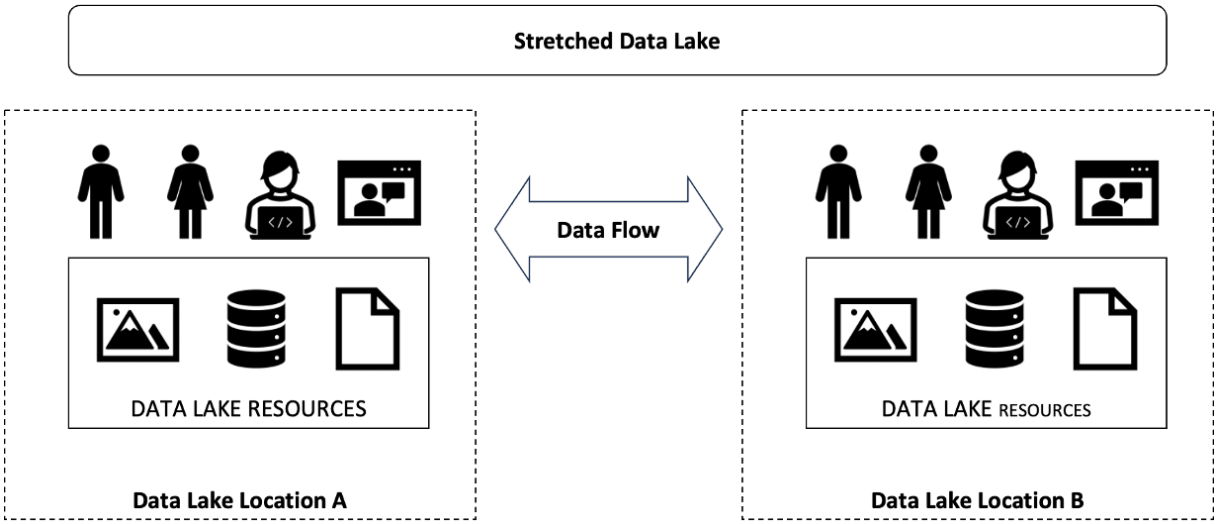

Figure 1. The Stretched Data Lake

The fact that data, processing, and pipelines are dispersed across multiple locations calls for a need to control, drive and manage processing in multiple environments in the continuum. More so, the control needs to address data governance (i.e., gravity and friction), which requires a need to understand the full flow of information and the appropriate restrictions on the use of data in each location.

In TEADAL the stretched data lake which is spanning multiple locations is managed by a unified and central mechanism enabling multi-location orchestration of data pipelines and data access.

The use of the TEADAL control plane, that manages the behavior of the data lake components, data pipelines and processing across all locations, addresses the complication of needing to control workloads in each location separately.

1.1 THE CHALLENGES OF IMPLEMENTING A STRETCHED DATA LAKE

The expansion of the data lake in TEADAL into a stretched data lake that spans multiple locations raises challenges that now need to be addressed as part of the data lake architecture. The challenges range from the skills of the data lake operators, the use of computational resources and regulatory and compliance requirements.

1.1.1 Management of multiple locations

The TEADAL stretched data lake spans multiple locations which requires the TEADAL software stack to run in multiple locations and serve data in these locations. This calls for the data lake operator to manage, configure, monitor, and operate the data lake resources in all the data lake locations.

In order to simplify the management of the TEADAL data lake we opted to adopt Kubernetes as the running platform. However, the data lake team still needs to control the data lake

Funded by the European Union

services and workloads that run in each cluster that participates in the stretched data lake. The result would be time consuming and tedious.

Our approach in TEADAL is to leverage a control plane that would act as a central management system and control the various workloads that run on all of the clusters that participate in the data lake.

Since TEADAL is a federation of data producers and data consumers there is a need to provide trust between producers and consumers of data. The trust aspects are detailed in a separate deliverable "D5.1 - Trustworthy Data Lakes Federation First Release Report". While this deliverable focuses on the orchestration aspects of multiple locations.

1.1.2 Optimizing the use of resources for achieving energy efficiency

The introduction of multiple locations adds new considerations for using data in the data lake such as the computing capabilities and costs in each location and the data movement constraints and costs for transferring data between the data lake locations.

The TEADAL data lake should optimize the use of resources and its environmental impact when managing the data lake. Specifically, it should optimize the overall cost (monetary, energy, etc) when providing data to consumers in the data lake.

1.1.3 Controlling the data across multiple locations

Support for a federation of multiple organizations each providing a data lake location, for either providing data, consuming data, or both, adds constraints on where data can be stored or processed.

The constraints stem from two main aspects:

1. **Regulatory concerns** that require removing or limiting the information that is transferred. For example, when transferring data between locations that cross country borders or regulatory domains.
2. **Data sharing agreements** that are formed between producers of data and consumers of data. The data sharing agreements determine which data is to be shared with an external consumer and define if locations provided by the data consumer can be used for storing or processing the data. Storing data in a consumer-provided location requires the data to adhere to the restrictions of the data sharing agreement, for example in the case that the consumer should receive only partial medical information, and not the whole medical dataset.

Observing data governance compliance should be embedded with the data processing in the data lake and be taken into account when deciding on the location of data processing in the stretched data lake.

1.2 DATA LAKE ARCHITECTURE IN TEADAL

The TEADAL data lake architecture aims to address the inherent challenges that the TEADAL approach poses. In this section we show the data lake design and how the architecture enables TEADAL data lake to address the challenges.

Funded by the European Union

A key feature of the TEADAL data lake is enabling storage, processing, and sharing of data across multiple locations and logical zones. Data can be stored and processed in each of the data lake locations and zones. This is enabled through the use of TEADAL control plane which conceal the fact that the data lake is composed of multiple locations and allow TEADAL software to run as-is without being aware of the actual running location.

In each location of the TEADAL data lake it is possible to have all logical zones.

Staging zone. A computation only zone that hosts TEADAL pipelines and processing that ingest data into the data lake from external sources. The staging zone is a computing zone, including transient storage, that spans compute resources across the data lake locations. These compute resources have access to external sources and can only read data from external systems (or transient data that is located in the staging zone) and write only to the curated zone. The staging zone does not host long term data, but only hosts the data that is needed as part of the processing, i.e., intermediate data.

Curated data zone. A storage zone that hosts data that is ingested into the data lake from external sources or data that was processed in the computation zone (see next). The curated zone is a storage zone only and does not provide processing capabilities beyond those available in the storage system themselves (e.g., SQL processing in a database). Data from the curated data zone is available for internal sharing in the data lake. That is, it can be accessed by processing that runs in the computation zone or data sharing zone. The data in this zone is properly classified and described in the data catalog. The curated zone can span multiple locations and can have available storage resources in all the data lake locations.

Computational zone. A computation only zone that hosts TEADAL pipelines and processing. This is where data analytics occurs within the data lake for the purpose of processing and refinements of the data that is stored in the curated data zone. In that case, the processing reads data from the curated data zone and writes the processed data back to the curated data zone. In addition, data can be prepared for sharing, in this case the output is written to the data sharing zone.

Data sharing zone. A storage and computation zone which focuses on enabling and implementing external data sharing. The zone hosts data which is a candidate to be shared and implements the sharing of data outside of the data lake, such data is referred to as a Federated Data Product (FDP) in TEADAL. The data sharing zone provides the compute and storage resources for implementing FDPs. The data sharing zone is available in all TEADAL data lake locations, and in locations that are provided by external consumers.

TEADAL pipelines are the mechanism in which data is transferred between zones and locations. When moving between zones, note that data can: (i) be read from the curated data zone and be written back to the curated data zone; and (ii) be read from the curated data zone and be written to the data sharing zone. Data is not allowed to be moved from the data sharing zone to the other zones.

When data is transferred between locations it still needs to follow the zone restrictions, and in addition the data pipeline needs to apply data friction concerns that relate to the fact that the location of the data is changing.

1.2.1 Federated Data Product

The Federated Data Product (FDP), see Figure 2 which appears in D3.1, provides the mechanism to serve data that is intended to be shared outside of the TEADAL data lake. The FDP is deployed in one of the locations of the data sharing zone, and can serve data that is

Funded by the European Union

physically located in the curated data zone or in the data sharing zone. The FDP is typically accessed by TEADAL pipelines to generate an instance of the data to be shared with an external consumer, see shared Federated Data Product in the next section. External data consumers cannot access the FDP in order to obtain data.

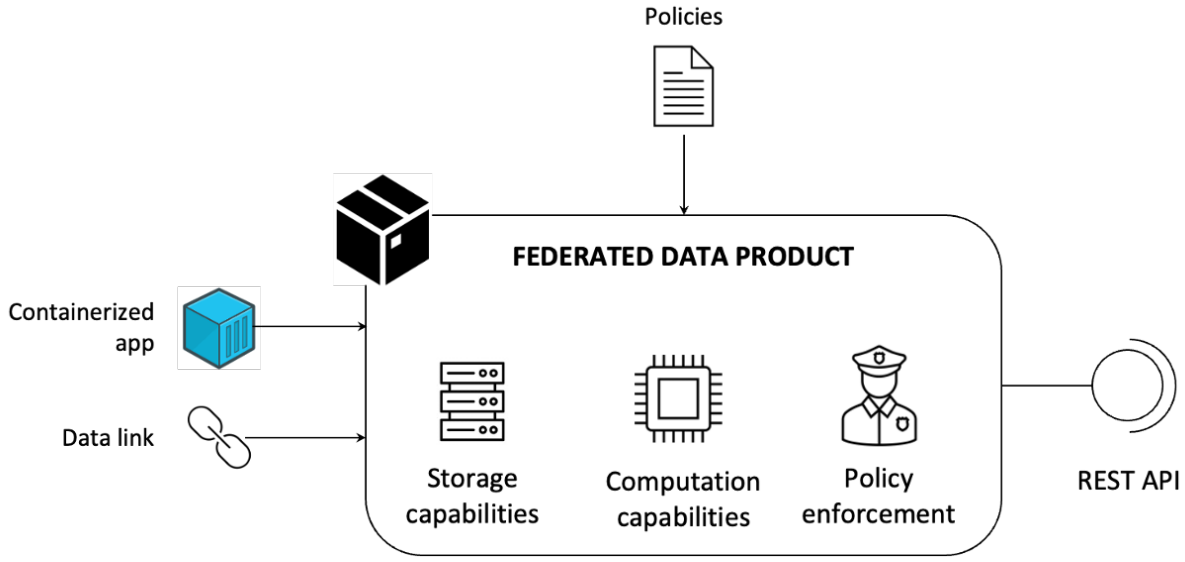

Figure 2. Federated Data Product

The FDP provides a REST API, deployed in the data sharing zone, that is accessed by internal applications and TEADAL pipelines that run in the data sharing zone. The FDP is responsible for implementing the access policies and delivering compliant data.

1.2.2 Shared Federated Data Product

The Shared Federated Data Product (SFDP), see Figure 3, provides the mechanism to serve data to external data consumers. The SFDP components are deployed in one of the locations of the data sharing zone. The SFDP construct is fed from existing FDPs either directly (linking) or indirectly through a TEADAL pipeline.

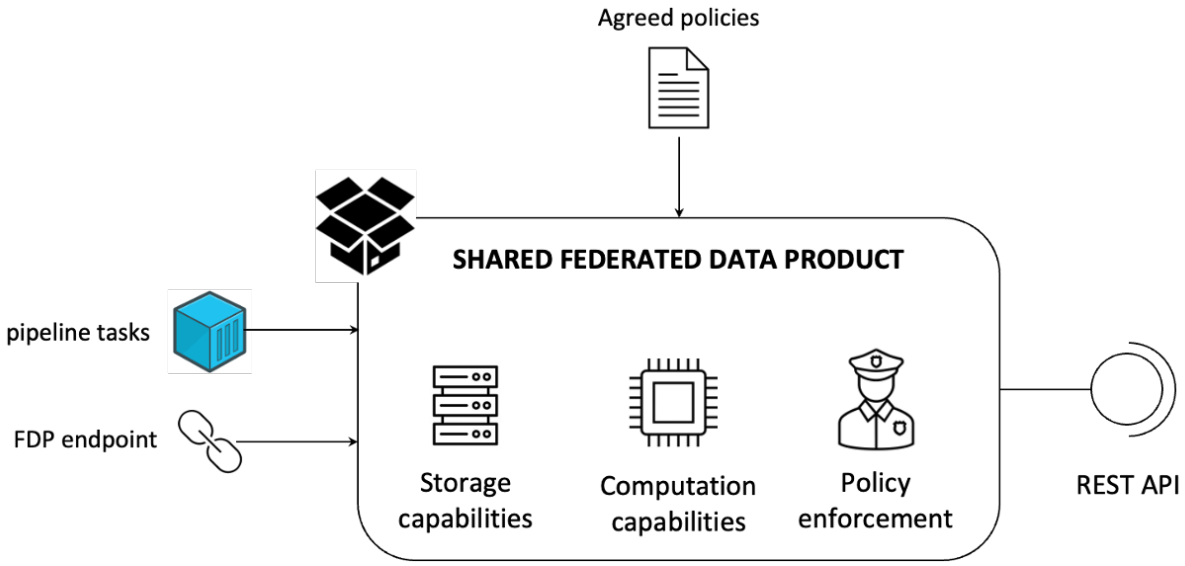

Figure 3. Shared Federated Data Product

Funded by the European Union

Similar to the FDP, the SFDP provides data through a REST API. In TEADAL initial release, when the SFDP is set up, the TEADAL pipeline is created to consume data from an existing FDP and produce a ready to be consumed dataset which is stored in the SFDP storage. The pipeline is deployed in one or more of the locations of the data sharing zone. When a consumer reads data from the SFDP it uses the REST API to consume the data.

1.2.3 TEADAL pipelines

TEADAL pipelines are the mechanism used to process, refine, or move data in the TEADAL data lake. The pipelines are authored by the creators of the FDPs and SFDPs. In the first release of the TEADAL solution we implement TEADAL pipelines to create a sFDP from a FDP manually. The FDP to sFDP pipeline runs in the data sharing zone, but can be deployed across multiple locations. The pipeline encodes all the processing to enforce that the agreed policies defined in the SFDP are implemented before storing the data in the sFDP storage. In Figure 4 we see the processing steps to transform and prepare the data read from the FDP REST API before writing the data to the sFDP storage.

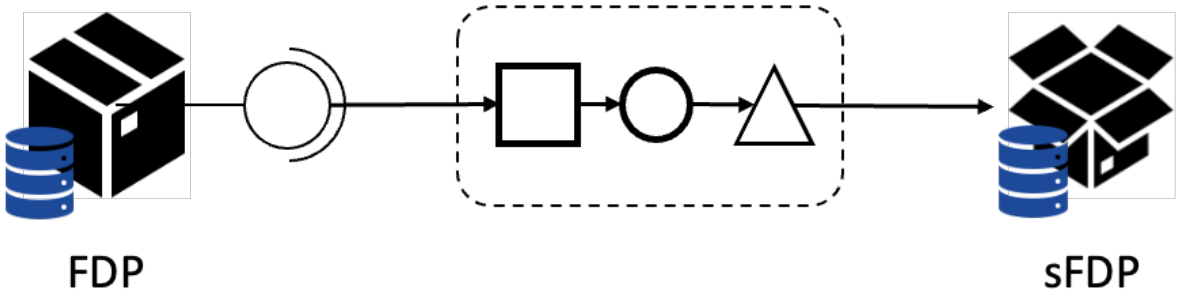

Figure 4. FDP to SFDP pipeline

The pipelines are designed so they can be optimized, taking into account the locations of the producer (FDP) and the data consumer (SFDP). For example, if the agreement calls for sharing only a subset of the data, then the filtering would be performed at the producer (FDP) location; or if the agreement calls for a specific formatting of the data, then the format processing can be performed at either location based on a cost optimization function.

1.2.4 TEADAL data lake control plane

The data lake control plane is responsible for providing a single control interface for running the data lake components (e.g., FDP, SFDP, pipelines). The goal of the control plane is to reduce the complexities of managing multiple locations. It enables the data lake team to run workloads using the control plane Kubernetes API, and the control plane is responsible to distribute and apply the workloads to the right cluster location.

The control plane itself does not determine the locations, but rather the location is provided through a logical mapping between the workload description and the Kubernetes cluster in the proper execution location. The logical mapping reflects both the availability of resources, i.e., the available resources in the Kubernetes cluster, and sharing agreements, such as the agreements between data provider and data consumer.

Funded by the European Union

2. TEADAL CONTROL PLANE

In TEADAL the data lake is extended to the cloud continuum, that is multiple locations (i.e., clusters). The control plane provides a mechanism to distribute and share across locations, and not be restricted to keeping data in one location. More so, in TEADAL we leverage a unified control plane no matter the underlying implementation or capabilities of the resources provided by a location --- be it edge location or cloud location.

The TEADAL data lake is running on top of Kubernetes clusters, this provides TEADAL with a simple well defined and industry acceptable interface to run the data lake workloads. The decision in TEADAL to expand into multiple clusters introduced the need to interact with multiple management endpoints for the clusters — as the Kubernetes control API is scoped to a single cluster.

In TEADAL we have chosen to control the workload across all clusters using a single management endpoint, and have decided to not introduce a new API, but rather to re-use the Kubernetes API for multiple locations. In order to achieve this goal, we opted to adopt Kubestellar[1], an open-source solution which is sponsored by IBM. Kubestellar provides a single workload management API for multiple clusters. Kubestellar provides a management endpoint allowing users to make use of the Kubernetes API and to be oblivious to the running location of the workload. It provides the impression of using a single Kubernetes cluster, but in fact acts as a mediator and distributes the workload to the intended locations.

The choice of adopting Kubernetes API for managing the data lake workloads enables TEADAL to leverage all existing Kubernetes tools (e.g., Helm[2], ArgoCD[3], …) natively without the need to add an additional integration layer for Kubernetes tools. Rather, it allows the use of Kubernetes labels and configuration to deploy workloads into multiple cluster locations. We expand on how Kubestellar simplifies using multiple locations later in the document. Note that in the first iteration TEADAL is implemented in one location, but the foundation is already in place for multiple locations.

The availability of multiple Kubestellar cluster locations across the cloud continuum introduces opportunities for decisions on which cluster to run, and what part of the workload should run in a given cluster location, for example, based on location capabilities (e.g. edge vs cloud), geographical location, or legal considerations. In TEADAL we use metadata to provide insight into the decisions to run in a given cluster. The metadata is associated with the running infrastructure (i.e., the cluster) and the data. Section 2.2 expands on collecting and generating metadata about the cluster health and performance in order to optimize the data lake workload. We expand on metadata for the data lake datasets in section 3.5.

In the first iteration we have explored and designed how to manage a workload across multiple Kubernetes clusters. In the TEADAL data lake we aim to partition TEADAL pipelines across multiple clusters. In the first iteration of the TEADAL data lake, the data lake is scoped to a single location. Hence, here we show the foundation which we have built for the following iterations.

[1] https://docs.kubestellar.io/main/
[2] https://helm.sh/
[3] https://argo-cd.readthedocs.io/en/stable/

Funded by the European Union

2.1 STRETCHED DATA LAKE WITH KUBESTELLAR

The TEADAL control plane is addressing the challenge of controlling the data lake workloads across multiple locations. Therefore, we have chosen Kubestellar to address the distribution of the data lake workloads. Kubestellar provides a central management point for deploying Kubernetes workloads (i.e., Kubernetes resources) which will in fact be executed on multiple clusters. The Kubernetes clusters on which workloads are actually running are called execution clusters or remote clusters. These clusters can run at edge locations, private cloud or public cloud. Kubestellar itself run in a Kubernetes environment, and can be provisioned in any available cluster. In TEADAL we will run Kubestellar in the Kubernetes cluster that will act as the control plane cluster where other common data lake services will run.

Kubestellar is based on and extends the Kubernetes API in a way that enables the users to leverage all existing tools that use the Kubernetes API — this enables TEADAL to natively support all existing data lake tools that are Kubernetes compatible.

In Figure 5 we describe how the TEADAL control plane leverages Kubestellar. In the figure we see three Kubernetes namespaces. These namespaces hold Kubernetes workload objects (e.g., config maps, deployments, services, …), which should be running on remote cluster locations. The TEADAL data lake cluster locations are described in the Kubestellar inventory (see next). In each location (e.g., location A) runs Kubestellar Syncer, which is responsible to deploy at the local location the workload objects that should run in the specific location. Next, placement rules are provided to encode which workload objects should be running in which location. We distinguish between two types of placements. The first where we define that all workload objects should be running in a given location, for example the workload objects in namespace A. The second is the case when the workload objects should be partitioned between locations, for example see the pipeline namespace.

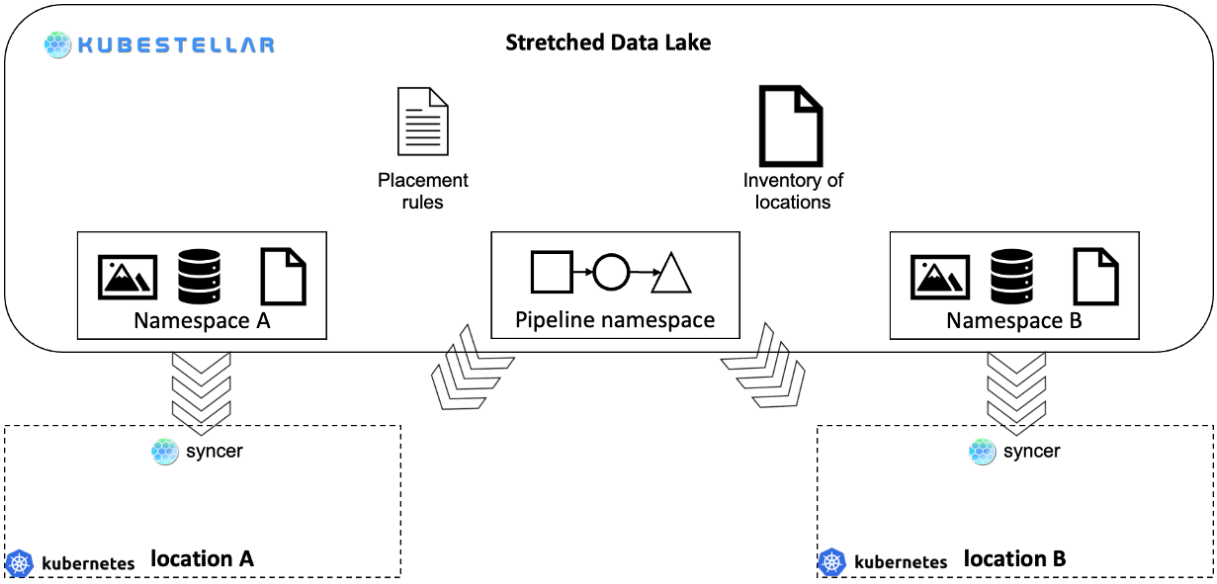

Figure 5. Applying Kubestellar for The TEADAL Data Lake across the continuum

Kubestellar provides a space abstraction, which is an HTTPS endpoint similar to the Kubernetes API servers, which can be accessed by standard Kubernetes client tools. The spaces are used to hold Kubernetes objects that facilitate the mapping to the actual execution clusters.

Funded by the European Union

Kubestellar adds (i) an inventory space that is used internally to store inventory objects that describe the remote clusters; and (ii) a workload description space that stores the workload objects and additional placement related objects.

The inventory space is used to store the description and details of the remote clusters. A cluster is added to the inventory using the Kubestellar API. In Figure 6 we see an example for adding a new cluster location to the inventory. Next the Kubestellar Syncer should be deployed on the remote cluster, enabling Kubernetes resources to be synced to the remote cluster.

```
kubectl kubestellar prep-for-cluster --imw root:teadal ks-teadal-cluster1 \
  env=ks-teadal-cluster1 \
  location-group=loc-teadal1
```

Figure 6. Example of adding location to Kubestellar

The workload description space is used to store the workload objects that would be running in the various locations and the placement rules, called `EdgePlacement` objects. This space is the unified view of the data lake workload. All of the data lake workload is deployed into this space. For example, definition of namespaces, deployments, config map, services, and more.

In Figure 7 we see an example for a placement rule (of type `EdgePlacement`) that specifies what Kubernetes objects should be applied in the target cluster location. The example defines an EdgePlacement named "`teadal-placement-locA`". The `locationSelectors` key specifies the cluster(s) that the mapping applies to. The "`downsync`" key specifies what resources are to be applied (synced) to the cluster that is labeled by `location-group:loc-teadal1`. In this example, all the configmaps and deployments objects from namespace `namespace-a` are selected to be synced.

```
apiVersion: edge.kubestellar.io/v2alpha1
kind: EdgePlacement
metadata:
  name: teadal-placement-locA
spec:
  locationSelectors:
  - matchLabels: {"location-group":"loc-teadal1"}
  downsync:
  - apiGroup: ""
    resources: [ configmaps ]
    namespaces: [ namespace-a ]
    objectNames: [ "*" ]
  - apiGroup: apps
    resources: [ deployments ]
    namespaces: [ namespace-a ]
    objectNames: [ "*" ]
  - apiGroup: apis.kcp.io
    resources: [ apibindings ]
    namespaceSelectors: []
    objectNames: [ "bind-kubernetes", "bind-apps" ]
```

Figure 7. Example for a placement rule that syncs resources to the target cluster location

Funded by the European Union

In Figure 8 we see an example workload that includes creating a namespace, named `namespace-a`, and deploying a config map and a web server as a sample for a generic deployment that is applied into the workload workspace. Deploying workload objects into `namespace-a` results in these objects (including the namespace if needed) to be deployed in the remote cluster(s) that match the location label.

```
apiVersion: v1
kind: Namespace
metadata:
  name: namespace-a
---
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: namespace-a
  name: httpd-htdocs
data:
  index.html: |
      ...
---
apiVersion: apps/v1
kind: Deployment
metadata:
  namespace: namespace-a
  name: web-server-deployment
spec:
  template:
    metadata:
      labels: {app: common}
    spec:
      containers:
      - name: httpd
        image: library/httpd:2.4
      ...
```

Figure 8. Example for a workload that is deployed into the Kubestellar workload space

The combination of deploying Kubernetes workload objects into “virtual” namespace and the placement rules defined by `EdgePlacement` mapping provides a simple mechanism to distribute the workload to multiple locations, and enables partitioning of a TEADAL pipeline based on optimization criteria such as availability and utilization of resources in a given cluster, data gravity or data friction.

The implementation of “distributing the workload” to the appropriate cluster is performed by the Kubestellar Syncer which is installed on the remote cluster. The Syncer is installed as part of adding a cluster to Kubestellar (see Figure 6). The Syncer looks for Kubernetes objects that should be synced to the remote cluster, and deploys those objects in the remote cluster. In addition, it reports back the status (i.e., the information in the status section of the Kubernetes objects) of the objects so information about the status of the deployment would be available to the user of Kubestellar.

Funded by the European Union

As the Syncer deploys the Kubernetes objects to the remote cluster, these objects are handled and managed by the local cluster mechanism as any other Kubernetes object. Any update to these objects (e.g., the status section) by local mechanism are shared back to the Kubestellar space by the Syncer.

2.2 OPTIMIZING THE DATA LAKE PLATFORM PERFORMANCE

Monitoring and collecting the state and performance metrics of the data lake resources is an integral part of the TEADAL control plane, specifically dealing with its project objective of delivering efficiency for building and using stretched data lakes solutions (see Obj1 in Project proposal, section 1.1.2).

A specific goal related to this objective can be translated into the goal of providing visibility of the status of the TEADAL data lake, and to introduce elements to support the automation of performance optimization.

This section introduces AI-driven components for performance monitoring of the TEADAL infrastructure and application environment. They will provide data and insights needed for the optimization of the data lake operations.

The following four elements of the data lake performance monitoring components are discussed.

- collecting measurable performance indicators
- dashboard-based real-time or near real-time performance visibility
- anomaly detection and predictive analysis
- alerting and notifications

The fact that the TEADAL data lake spans multiple locations introduces complexities and opportunities to optimize the distribution and sizing of resources across locations. The insight produced by the AI-driven components, whilst enabling the application of performance analytics, will also be used to feed an “actuator” that will take appropriate actions to optimize performance in the data lake.

Performance monitoring of the TEADAL data lake is the systematic process, especially concerning and involving data lake operators, of observing, measuring, and analyzing various aspects of the platform operations and infrastructure to ensure its efficiency, reliability, and optimal usage.

Figure 9 below shows how such AI-based components, described in more detail in the following sections, fit into the general TEADAL data lake monitoring infrastructure. They are currently in the design and experimentation phases and will be implemented in the next project iteration.

Funded by the European Union

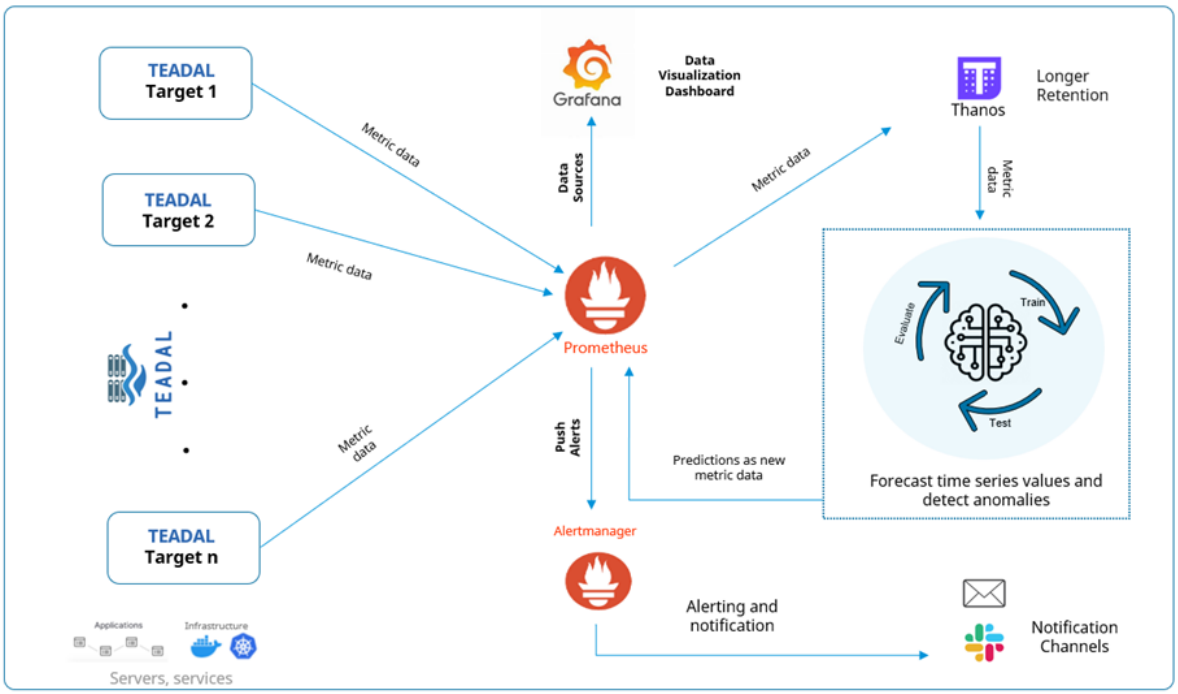

Figure 9. AI-driven performance monitoring of TEADAL data lake

2.2.1 Indicator metrics collection

Performance indicators of the data lake platform will be collected using a pull-based model, based on the open-source Prometheus[4] tool, that periodically queries the data lake layer targets (applications, services, servers) to gather metrics. The collection of the indicator values requires the specification of data lake targets by defining the hostname, port, and the endpoint path (typically "/metrics") where the targets expose their metrics data.

After being configured with target endpoints and scrapping intervals, Prometheus periodically sends HTTP/HTTPS requests to these endpoints, requesting the metrics information. The targets respond to these requests by providing the current metrics data in an agreed format, usually structured plain text or a particular JSON schema.

This data includes key-value pairs representing different metrics, such as CPU usage, memory consumption, and other performance-related indicators, listed in Table 1 below.

[4] https://prometheus.io/

Funded by the European Union

Table 1. Candidate performance metrics for monitoring TEADAL data lake

Category	Metrics	Data source
System Performance	CPU Utilization, Memory Usage, Disk I/O, Network Throughput, Storage Utilization, Node Availability, Latency	Metrics
Data Processing	FDP Response Time	Metrics
User Experience	Response Time, Dashboard Loading Time, Data Visualization Rendering Time	Metrics/Logs
Resource Utilization	Node Resource Allocation, Resource Contention, Scaling Efficiency, Storage Efficiency	Metrics
Cluster & Node Level	Cluster Resource Utilization, Cluster Availability, Node Health, Node Connectivity, Node Storage Utilization	Metrics/Logs
Service & Partition Level	Service Availability, Partition Size, Service Response Time, Partition Throughput, Service Errors	Metrics/Logs
Storage Level	Disk Space Utilization, I/O Operations, Storage Health, Storage Latency	Metrics/Logs
User & Pipeline Level	User-specific FDP Performance, User Activity Patterns, Pipeline Execution Time, Pipeline Success Rate	Metrics/Logs
Task & Database Level	Task Execution Time, Task Resource Utilization, Read/write Performance per dataset/table, Database Size	Metrics/Logs

In Prometheus, specific metrics can be custom defined, for example including data lake performance indicators. Additionally, metrics often come with timestamps indicating the time of data collection identified by metric name and key/value pairs, allowing Prometheus to maintain accurate time series data for analysis and alerting purposes. Moreover, metadata like labels can be included, offering contextual information about the metrics, and enhancing Prometheus ability to organize and query the data effectively.

After collecting the various metrics, they will be persisted in a long-term retention repository by using the open-source tool Thanos[5].

[5] https://github.com/thanos-io/

Funded by the European Union

2.2.2 Real-time performance visibility

The open-source Grafana[6] tool provides real-time visibility of insights and monitoring capabilities within the TEADAL data lake. It is a platform that enables the creation of dynamic and interactive dashboards to visualize real-time metrics and data.

It will be used mainly as an integration and visualization tool for data sources gathered by Prometheus, where AI developer, members of the Data Lake Operators' teams, are allowed to track key performance indicators, system health, and usage patterns in real time. In addition, the Grafana features will be leveraged for our custom AI solution in assisting ML development and alert management.

2.2.2.1 Toward AI-driven performance monitoring

AI-driven performance issue identification and insight in the TEADAL platform involve (1) anomaly detection and (2) predictive analysis by using AI techniques. This is a custom service that will be developed during the project and that will be added to the main monitoring tool, Prometheus, through configuration instrumentation via Python client libraries for Prometheus.

While Prometheus is a versatile performance monitoring tool for distributed systems, its lack of native AI-driven capabilities requires careful consideration and additional efforts to implement AI-driven monitoring in its workflows.

The process of TEADAL's AI-driven monitoring begins with gathering extensive metrics and performance data. These metrics encompass critical information, including CPU usage, memory consumption, network traffic, and other performance-related indicators. While Prometheus excels in real-time monitoring, AI-driven monitoring requires a reliable, scalable, and long-term storage solution for aggregated historical data. To address this need, as per established best practice, Prometheus is configured to use the Thanos open-source database tool. Once the metrics have been accurately aggregated and persisted in long-term retention repositories, they become valuable for AI-driven predictive analysis and anomaly detection in the platform.

Then the data stored in Thanos will be pre-processed to handle missing values, outliers, and noise, ensuring the quality of the dataset for future ML analysis.

ML algorithms require careful selection and engineering of features that best represent the performance behavior of the data lake platform. These features can be derived from raw metrics or created based on domain knowledge and play a vital role in improving the accuracy of anomaly detection and predictive analysis AI models.

2.2.2.2 Anomaly detection

This is the first AI model that will be implemented during the project activities. Anomaly detection is essentially a binary classification problem, where the classes are either normal or abnormal. Based on the data lake historical performance metrics and their trends, deviations from the expected behavior will be identified.

Anomalies could indicate sudden spikes in user activity or resource usage in the platform. Different machine learning algorithms, such as supervised learning (e.g., decision trees, random forests), unsupervised learning (e.g., clustering), or deep learning (e.g., neural

[6] https://grafana.com

Funded by the European Union

networks), can be used for anomaly detection. The selected machine learning model is trained on historical data stored in Thanos to learn normal patterns of the data lake platform performance. After training, the model is tested on separate datasets to evaluate its performance and accuracy. Evaluation metrics such as precision, recall, and F1-score will be used to assess the model's effectiveness in detecting anomalies.

The AI models, developed and validated through the previous steps, will be integrated into Prometheus Alertmanager (a Prometheus module) and Grafana using custom exporters or APIs providing a comprehensive view of the TEADAL's platform behavior and performance.

The detected anomalies promptly trigger alerts that will be notified to a future Control Plane "Actuator" component, focused to take informed decisions and actions in order to ensure the maintenance of optimal TEADAL data lake performance. The "Actuator" design and realization will be addressed in the second iteration of the project.

2.2.2.3 Predictive analysis

The second AI component that will be realized is the Predictive Analysis model, where the data lake performance indicator metrics scraped by Prometheus from the TEADAL target layer HTTP endpoints are ideally suited for time-series predictions due to their inherently structured and timestamped nature. By having Prometheus metrics captured at regular intervals, forming a coherent time-ordered dataset, we can implement a predictive analysis for understanding trends, patterns, and seasonality within the data.

Machine learning models, particularly designed for time-series forecasting like ARIMA (AutoRegressive Integrated Moving Average) and LSTM (Long Short-Term Memory), thrive on such sequential data. They can exploit the historical patterns and correlations present in Prometheus metrics to make accurate predictions about future TEADAL platform behavior.

The AI-based predictive analysis model that will be developed, is expected to provide to the same future Control Plane "Actuator" prediction results for insights on the anticipated demands, resource usage patterns, seasonalities-based on historical trends and future predictions, this is insightful for early issue detection, facilitating prompt resolution, and improving TEADAL system reliability.

2.2.3 Alerting and notification

The Prometheus Alertmanager[7] powers alerting and notification systems in the TEADAL platform. These monitoring mechanisms signal the need for immediate attention to the platform highlighting potential issues before they escalate into critical problems.

The process of alerting and notification relies on several key components. First, Alert Rules are configured based on a specific set of conditions and thresholds. When these conditions are met or breached, the alert engine evaluates the rules and triggers Alert Firing. Before notifications are sent out, alerts can be customized using predefined templates through Alert Templating.

Firing alerts are sent to Prometheus Alertmanager where they undergo Alert Deduplication and Grouping for efficient handling. Routing Configuration defines the rules to determine how alerts

[7] https://prometheus.io/docs/alerting/latest/alertmanager/

Funded by the European Union

should be directed, and notifications are sent to designated channels such as email, SMS, Slack, or webhooks through Notification Integration.

So, in the context of the service assurance process, the “Actuator” can receive and acknowledge notifications coming directly from Prometheus. The acknowledgment prevents further notifications for the same issue from being sent by Alertmanager. When the underlying problem is resolved, alerts transition to a resolved state, completing the monitoring and alerting cycle.

2.2.4 Implementation status

After an initial feasibility study based on an in-depth literature review on the definition of technical requirements about the metrics to be produced by the observed TEADAL components, the current activities are in the design and experimentation phases. We report here the current outline of the software tools serving the realization of the performance monitoring solution.

The open-source tools Prometheus and Grafana will be leveraged for monitoring the TEADAL data lake. The time-series dataset from Prometheus will be stored in the long-term open-source storage tool Thanos to assure the persistence of historical data.

In addition, the functionalities of the Prometheus/Grafana-based observability architecture will be extended during the project activities with the design, implementation, and integration of two new AI-driven models trained and tested based on the collected historical monitoring data. The two ML models are:

(i) Anomaly Detection AI model: to analyze time-series data and to identify anomalies or irregularities of performance issues

(ii) Predictive Analysis AI model: to implement predictive analytics to forecast future resource utilization or potential issues.

As earlier described, this AI-driven approach will help to provide visibility of the status of the TEADAL data lake, and will contribute elements ultimately necessary for the automation of performance optimization.

As anticipated early, at the time of writing this document, these AI models are under design and experimentation. They will be implemented in the next project iteration.

Funded by the European Union

3. MANAGING FEDERATED DATA PRODUCTS

Federated data products are the mechanism that TEADAL offers for sharing data with external consumers. A Federated Data Product is explicitly created and becomes a candidate for sharing with external parties. Creating a Federated Data Product results with a deployment of compute artifacts that provide controlled access to data. However, the data is not externally visible yet, but is available in the data catalog, and can be selected for consumption by an external consumer.

Enabling the actual sharing is performed by creating a Shared Federated Data Product, which is created as a result of an agreement between a product owner and an external consumer. At the time of creating the shared data product, a TEADAL pipeline, which is responsible for creating a sharable version of the data is created, and an access mechanism, responsible to provide the data to the consumer, is deployed.

In the next sections we describe the foundation of data sharing in TEADAL: (i) the data catalog which is used to hold the descriptions of the FDPs (and SFDP), (ii) the technical process of creating a FDP (and SFDP), and (iii) the access control mechanism for FDPs (and SFDPs).

3.1 TEADAL DATA CATALOG

The first release of the TEADAL catalog is a container-based application, comprising several components (as shown in Figure 10 below) which are configured and started using Docker Compose. In the first iteration of the TEADAL development process, the catalog is intended to be a centralized component, shared by all members of the federation. As such, it has been deployed on a server owned by Cefriel as a Docker Compose application. The Docker Compose descriptor defines the deployment of the entire stack, so that it is possible to start all the components with a single command.

The TEADAL catalog can also be configured to connect to already existing servers implementing specific features, as in the case where a Jenkins[8] server or an RDF Repository are already running in the deployment environment. The TEADAL catalog has been implemented by integrating several open-source software, all available with non-viral licenses, following the principle of not reinventing the wheel when dealing with specific topics (like DevOps or workflow management) and reusing what the open-source community already offers.

During the second iteration, we will focus on integrating the catalog in the TEADAL base node installation. This will require:

- Deployment on Kubernetes environment via ArgoCD pipelines
- Integration with the Identity Provider of the base node (Keycloak[9])
- Integration with the Object storage (Minio[10]) already provided by the base node

This will open the possibility to deploy one catalog per member of a federation, and will lead to supporting fully peer to peer scenarios where companies participating in data sharing federations will not be forced to maintain shared hardware or software resources.

[8] https://www.jenkins.io/
[9] https://www.keycloak.org/
[10] https://min.io/

Funded by the European Union

3.1.1 Catalog components

In the following, we will describe the main features of each component of the first iteration of the TEADAL catalog.

Figure 10. The TEADAL catalog architecture

Reverse proxy: The Reverse proxy component provides external access to the TEADAL catalog, routing the requests to the Web applications which are not directly exposed. Nginx[11] has been chosen for the purpose for its wide support and easy configuration of the secure communication layer.

Catalog API: The main TEADAL catalog API provides support for managing digital artifacts descriptions and their lifecycle. Moreover, it allows defining users, groups, and their associated privileges. This component have been implemented using Django[12] and Django REST framework.

Long-running processes: Since a Django application is not multi-threaded, long running processes (like mass mailing, or heavy I/O operations) must be handled in a separate component to avoid blocking the whole application until the request execution has been completed. To this extent, using a task queue like Celery[13] allows splitting the business logic of the application in two separate layers. All computationally hard or I/O-bound operations are implemented as tasks which are executed asynchronously. The Celery task queue sends such tasks to a pool of “workers”, which are separate processes on the same physical machine or on dedicated machines. The number of such workers is configurable using the Docker Compose configuration.

[11] https://www.nginx.com/
[12] https://www.djangoproject.com/
[13] https://docs.celeryq.dev/

Funded by the European Union

Caching: Redis[14] is a fast in-memory No-SQL database. In our deployment, it is internally used both as cache server and as communication middleware to off-load long-running tasks to the dedicated component.

Lifecycle management: The definition of a clear and widely accepted process to manage the lifecycle of an asset is a key element in promoting a sharing ecosystem. In the TEADAL catalog, we adopted the BPMN formalism to let ecosystem designer define such processes since it is widely used and provides a graphical representation which can be unequivocally interpreted by both managers and technical users. Each BPMN lifecycle defines the human and service tasks which must be triggered whenever a new asset is published into the catalog. Since the TEADAL catalog is able to handle multiple "digital artifact types", we let ecosystem designers attach a BPMN lifecycle to each asset type, so that it becomes possible to define different tasks related to different asset types. To implement this feature, we decided to integrate Camunda, an open-source BPMN engine. The component is started in its own container, and the integration takes place via HTTP calls, so that in principle it is possible to substitute it with another BPMN engine with minimal integration effort.

DevOps automation: The lifecycle of a specific digital artifact type can rely on the orchestration of different human and service tasks. With the BPMN process engine, it is possible to define a high-level complex process, but there are low-level tasks which cannot be easily specified using BPMN. Such low-level tasks, like creating deployable artifacts starting from an artifact description, or automatically deploying components onto a runtime environment, require the possibility to run scripts and interact with the filesystem. Those features are commonly implemented in Continuous Integration and Continuous Deployment (CI/CD) systems, and therefore we decided to integrate an open-source CI/CD solution (Jenkins) inside the TEADAL catalog. The Jenkins server is started in its own container and is integrated via HTTP calls, so that in principle it is possible to either integrate the TEADAL catalog with an already deployed Jenkins instance or to integrate with another CI/CD product with minimal effort.

RDF Repository: The TEADAL catalog uses an RDF repository to store the metadata of the assets whose publication has been approved. The metadata vocabulary is built upon DCAT[15] and ADMS[16] specifications, therefore reusing best practices for catalog interoperability. Blazegraph[17] has been chosen for the purpose, but in principle, any product implementing SPARQL[18] and the SPARQL Graph Protocol can be used.

3.1.2 Technical Metadata

Technical metadata describes characteristics about datasets that are used by the Stretched Data Lake Compiler (SDLC) to reduce the impact of gravity and friction in data pipelines. The exact metadata served by the technical metadata catalog will depend on the optimizations

[14] https://redis.io/
[15] https://www.w3.org/TR/vocab-dcat-3/
[16] https://www.w3.org/TR/vocab-adms/
[17] https://blazegraph.com/
[18] https://www.w3.org/TR/sparql11-query/

Funded by the European Union

implemented by the SDLC. We can consider the Parquet File Metadata[19] as a limited example of the types of metadata the SDLC will consume. The exact set of metadata attributes will be refined as the optimization techniques in the SDLC are developed. At the moment, we expect a need for schema information, dataset sizes (uncompressed, on-disk), column value distributions (when applicable), number of unique values in a column (when applicable), and other such metadata that are commonly used by database query optimizers. We may add dataset statistics that have not yet been proposed by existing database query optimizers.

For this deliverable, we will provide relevant technical metadata statically (e.g., in the form of YAML or JSON files) that can be fed as input to the SDLC. In later revisions, we plan to replace this with an open-source metadata catalog (e.g., OpenMetadata[20], Apache Project Nessie[21], Apache Hive Metastore[22]).

3.2 DEPLOYING FEDERATED DATA PRODUCTS

Federated Data Products are responsible for delivering data for sharing. The control plane is responsible for the actual deployment of the compute resources that support the FDP and SFDP.

The FDP and SFDP are packaged as Kubernetes deployment objects (e.g., Helm), and as described in Deliverable “D2.2 - Pilot cases intermediate description and initial architecture of the platform”, are packaged specifically for each data product. For details about the process of creating a FDP see section 9.4.3 and for creating a SFDP see 9.4.4 in D2.2.

The deployment objects are deployed to the TEADAL workload space, and are delivered to the remote cluster location that is associated either with the data producer deployment (for FDP) or with the data consumer (for SFDP).

In addition, in the case of deploying a SFDP, a TEADAL pipeline will also be deployed. The pipeline is responsible for preparing the data to be shared. The TEADAL pipeline implements the agreement that was reached between the data producer and data consumer. The pipeline is deployed to the TEADAL workload space, into the pipeline’s namespace. The pipeline objects are to be deployed to either the data producer location or to the data consumer location. We note that in the first iteration, TEADAL is actually implemented with a single location. In the next iteration the deployment will support multiple locations.

The details of how a TEADAL pipeline is implemented is described in Section 4.

3.3 ACCESSING FEDERATED DATA PRODUCTS

Deliverable “D2.2 - Pilot cases intermediate description and initial architecture of the platform” describes the security conceptual architecture. In a nutshell, communication between consumers and Federated Data Product services is mediated by interception proxies that interpose access control. Proxies delegate access control decisions and enforcement to a policy decision point and policy enforcement point, respectively. Multiple proxies could be deployed even for a single data product, depending on the number of service replicas for that

[19] https://parquet.apache.org/docs/file-format/metadata/
[20] https://open-metadata.org/
[21] https://projectnessie.org/
[22] https://cwiki.apache.org/confluence/display/hive/design#Design-Metastore

Funded by the European Union

product. Data Providers specify access control policies to govern how their data products ought to be consumed. A policy store makes these policies available to the policy decision point for evaluation against consumer requests to perform operations on data products.

The UML class in Figure 11 below illustrates all these security conceptual components and actors as well as their relationships. Note the role of the proxy: it acts as a mediator between the consumer and the data product itself, but allowing the consumer to interact with it as if it was the data product. From the consumer perspective, they are just communicating with the data product directly, the policy decision and enforcement flow concealed behind the proxy.

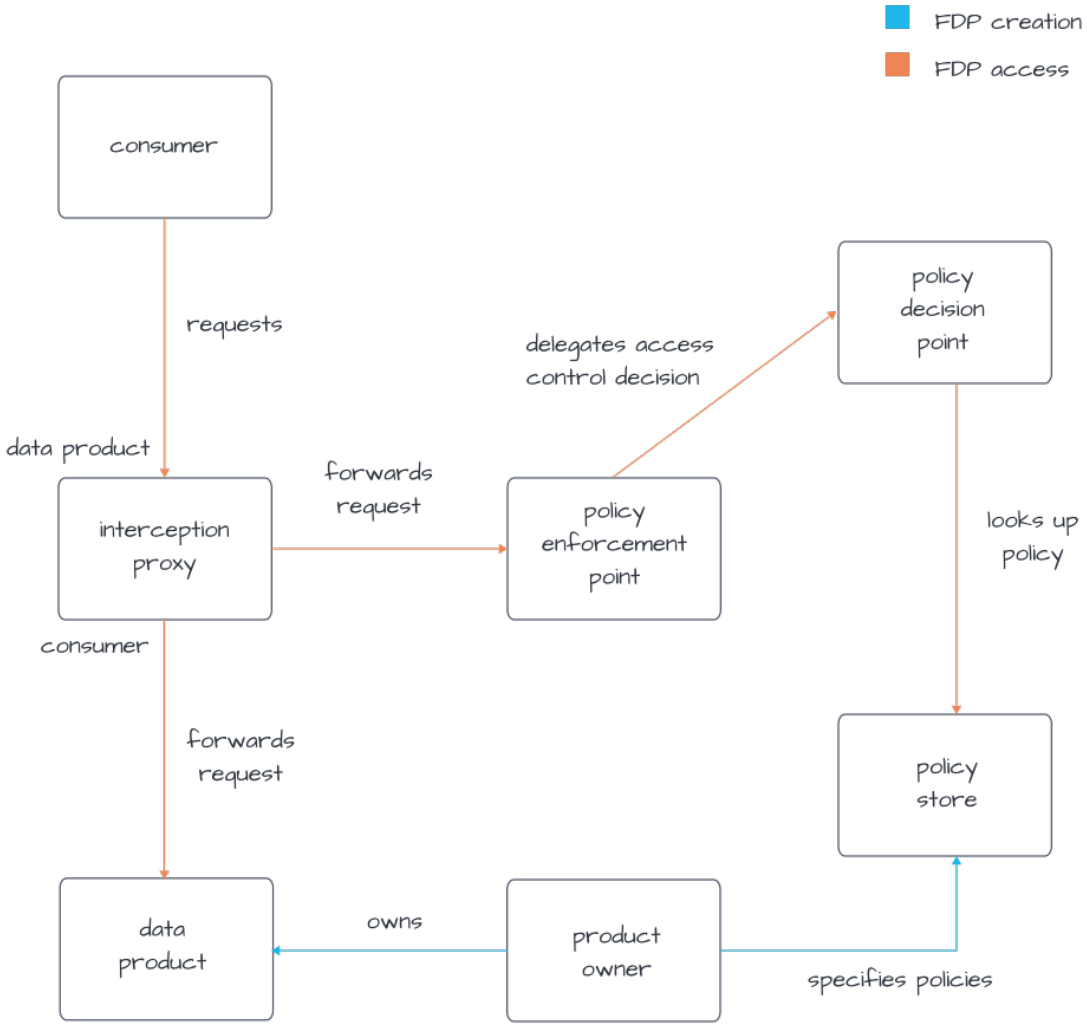

Figure 11. Security conceptual components and actors.

This section delves into the technical design and technology stack adopted to realize the aforementioned conceptual architecture. Note that the TEADAL security implementation embraces open-source: all the technology stack components, both third-party and TEADAL's own, are open-source.

Funded by the European Union

3.3.1 Message interception and access control delegation

Istio[23] provides the message interception facility. Indeed, Istio provides the infrastructure on which the control and data plane of the TEADAL data mesh are built. In particular, the data plane contains a network of proxies through which data product inbound and outbound HTTP traffic flow. A Federated Data Product service (FDP, SFDP) exposes a REST API. Consumers access the data product by making HTTP requests to this data product API. The Istio data plane proxies intercept each consumer request and data product service response.

In the TEADAL implementation, the control plane configures and deploys data product proxies on behalf of the data owner in the proper location. These proxies are Istio extensions of the Envoy[24] proxy tailored to work seamlessly with the Istio mesh. Istio pairs a proxy with each data product service upon service deployment to the mesh. As part of the deployment procedure, Envoy configures iptables rules (essentially ip packet filter rules of the Linux firewall) to make the proxy receive any IP packet destined for or originating from the data product service. Thus, the service is isolated from the network, in the sense that it cannot communicate directly with other mesh services or external clients as all IP traffic is forced through the proxy.

Envoy implements a pipes and filters design to process HTTP traffic. Each filter is a module handling a specific aspect of a request/reply exchange and filters are combined together in a request processing pipeline so that the output of a filter becomes the input of the next filter in the pipeline. The TEADAL implementation configures the Istio Envoy with an External Authorization Filter so that, on receiving a data product service request, Envoy triggers the filter execution before forwarding the request to the data product service.

The Authorization Filter acts as a policy enforcement point. It connects to an external policy decision point to determine whether to allow or deny data product service requests. The decision point can be any gRPC service implementing the External Authorization interface defined by the following Protocol Buffers specification

```
service Authorization {
  rpc Check(CheckRequest) returns (CheckResponse) {}
}
```

Figure 12. Authorization Filter

The filter invokes the Check procedure with a structure (CheckRequest) holding HTTP request data and additional Envoy parameters. The policy decision point responds either with an "allow" or "deny" decision, encoded in the CheckResponse structure. In the case of an "allow" decision, the filter lets Envoy forward the consumer's request to the data product service. On the other hand, in the case of a "deny" decision, the filter stops the request pipeline execution and instructs Envoy to return an HTTP Forbidden (403) response to the consumer instead of forwarding the request to the data product service.

[23] https://istio.io/latest/docs/ops/deployment/architecture/
[24] https://www.envoyproxy.io/

Funded by the European Union

3.3.2 Policy decision point and store

The Open Policy Agent (OPA)[25] is a key technology of the TEADAL access control implementation. OPA's Datalog-inspired Rego programming language affords policy writers the means to express access control rules as high-level, declarative logic queries on data product service requests. OPA's versatile runtime caters for evaluating queries against arbitrary JSON inputs 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.

The TEADAL Data Lake employs the OPA Envoy Plug-in as a policy decision point. This plug-in embeds the OPA server runtime and implements the External Authorization gRPC interface detailed earlier by evaluating Rego policies against the HTTP request dispatched by the External Authorization Filter. The TEADAL implementation connects the Envoy to the plug-in and configures the latter to fetch policy bundles from a dedicated Nginx server which acts as the policy store. The plug-in and Nginx are configured to use HTTP long polling so that the plug-in can efficiently update previously downloaded and cached bundles. After downloading a bundle, the plug-in also verifies its digital signature before evaluating the contained code to ensure that only trusted policies are enforced.

3.3.3 Interaction mechanics

The process of augmenting data product functionality with access control is as follows. First off, the Istio Envoy proxy intercepts the HTTP request which the data consumer makes to the SFDP API. Envoy handles the request following its configured request processing pipeline, which, as explained earlier, makes use of the TEADAL External Authorization Filter. The filter begins to process the request by invoking the configured gRPC server implementation of the External Authorization interface, that is, it calls the Check procedure passing in request data and other Envoy-specific parameters. If the Check procedure returns an "allow" decision, the filter makes Envoy route the original consumer's HTTP request to the data product service, collect the response and forward it to the consumer. On the other hand, if the Check procedure returns a "deny" decision, the filter causes the request pipeline to halt and makes Envoy return an HTTP Forbidden (403) response to the consumer. In this case the original consumer's request never reaches the data product service.

The OPA Envoy Plug-in is the configured service implementing the External Authorization interface. When the filter calls the Check procedure, the plug-in service retrieves the current Rego policies from the configured Nginx Web server. The plug-in caches and updates policies through HTTP long polling, as explained earlier. Having the current policies, the plug-in proceeds to evaluate them against the HTTP request data contained in the CheckRequest input to the Check call. These policies actually determine whether the request should be allowed or access to the data product service API should be denied. The plug-in puts the policy evaluation outcome in the CheckResponse structure and returns it to the filter.

The following Figure 13 illustrates the process described above for the case of an "allow" decision and summarizes the access control implementation discussion so far.

[25] https://www.openpolicyagent.org/

Funded by the European Union

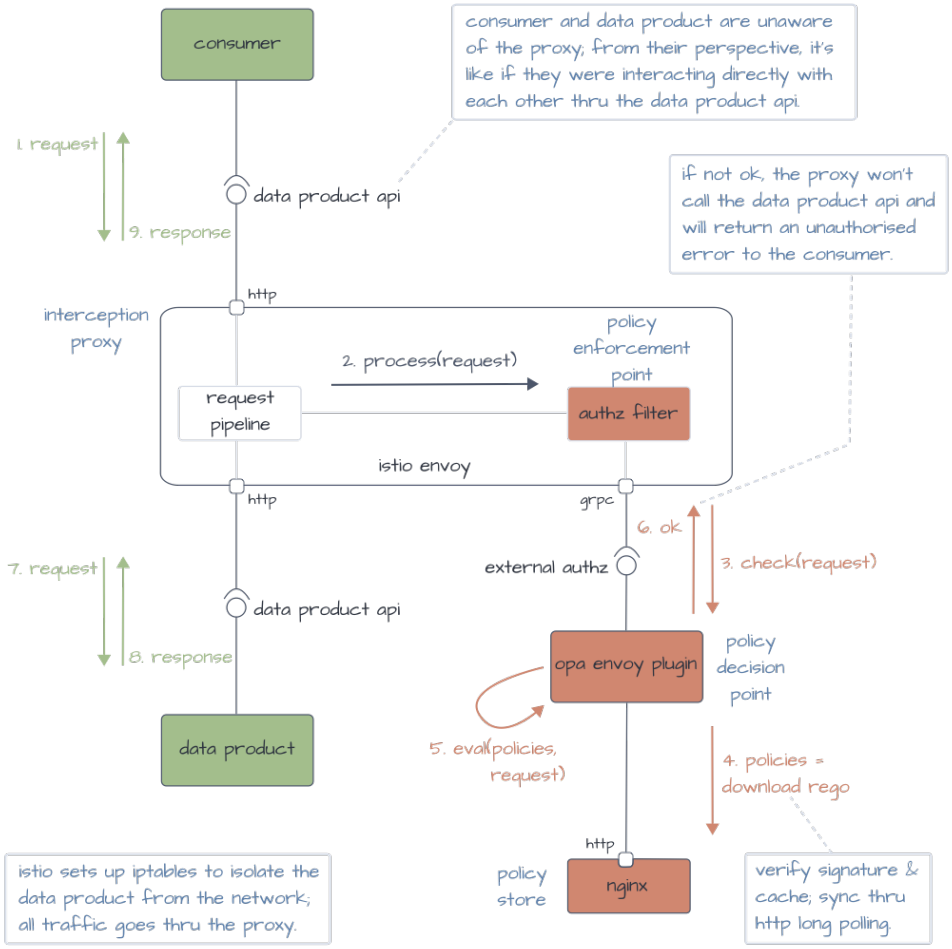

Figure 13. Example for "allow" processing.

Note that both the data product and the consumer are unaware of the Istio Envoy proxy. From the consumer's perspective, the HTTP request is a direct message to the data product API as if the consumer were invoking the API without a proxy in between. Similarly, the data product API processes the request as if it originated directly from the consumer and produces the same response it would if the proxy did not intercept the incoming request.

3.3.4 RBAC framework

While the machinery described so far can be used to enforce any kind of access control, TEADAL also provides a built-in Role-Based Access Control (RBAC) framework. This framework dramatically reduces the effort needed to implement access control for RESTful services, while still leaving policy writers the freedom to extend the base framework with service-specific functionality.

Data lake users are managed through a federated, OIDC-compliant identity management (IdM) service. Consumer services act on behalf of users who have proved their identity through IdM-configured procedures such as credential challenges, multi-factor authentication, etc. Upon successful authentication, the IdM issues an identity token, more specifically a JSON Web Token (JWT), which certifies the user's identity. Consumer services attach the token to each data product service request by means of the Bearer HTTP Authorization header. Presently, TEADAL deploys Keycloak as an IdM service, although any other OIDC-compliant software could be used too as the RBAC framework only requires OIDC-compliancy, making no assumption about the actual IdM implementation.

Funded by the European Union

RBAC roles, users and policy rules are written in plain Rego. Thus, policy writers are empowered with a fully-fledged programming language which they can exploit to customise, abstract and reuse their roles and policies to an extent that is simply not possible with traditional, configuration-based, cloud Identity and Access Management solutions. Moreover, policy writers can implement automated Rego tests to verify their policies have the desired effect when evaluated or even do that interactively, for rapid prototyping, as the OPA runtime has both test and read-eval-print loop (REPL) facilities. Extensive, automated tests also prevent regression issues where modifying a rule may have an unforeseen, unwanted side-effect, possibly leading to a security incident. Again, this level of sophistication is extremely expensive, in terms of required effort, to attain with traditional Identity and Access Management solutions.

The TEADAL authnz Rego library is a good case in point. Policy writers import this library in their code to automatically handle the evaluation of RBAC rules, user authentication, JWT validation, OIDC discovery as well as cryptographic keys download, verification and caching. The library allows policy writers to concentrate on defining their own, service-specific access control rules using an intuitive format.

By way of example, consider securing a simple FDP. The REST service exposes patient records as Web resources. There are three paths: /patients to list and add patients, /patients/id/ to retrieve and delete a particular patient, and /patients/age to retrieve a list with the ID and age of each patient but nothing else. Also, there's a /status path which returns the current service status. We would like to define two roles. A product owner, which should be able to perform a GET, POST and DELETE on any URL path starting with /patients, and a product consumer, which should only be allowed to GET patient ages and service status. Moreover, we would like to assign both the product owner and consumer roles to the user identified by the email of jeejee@teadal.eu whereas just the product consumer role to the user identified by the email of sebs@teadal.eu. In the TEADAL RBAC framework, all of the above can be accomplished with the following Rego code.

Funded by
the European Union

```
# Role defs.
product_owner := "product_owner"
product_consumer := "product_consumer"
# Map each role to a list of permission objects.
# Each permission object specifies a set of allowed HTTP methods for
# the Web resources identified by the URLs matching the given regex.
role_to_perms := {
    product_owner: [
        {
            "methods": [ "GET", "POST", "DELETE" ],
            "url_regex": "^/patients/.*"
        }
    ],
    product_consumer: [
        {
            "methods": [ "GET" ],
            "url_regex": "^/patients/age$"
        },
        {
            "methods": [ "GET" ],
            "url_regex": "^/status$"
        }
    ]
}
# Map each user to their roles.
user_to_roles := {
    "jeejee@teadal.eu": [ product_owner, product_consumer ],
    "sebs@teadal.eu": [ product_consumer ]
}
```

Figure 14. Rego policy for mapping roles to permissions

To evaluate our RBAC rules against the request received from Envoy, we would simply import the TEADAL authnz library and call its allow function as exemplified by the Rego code snippet below, where we tacitly assume the RBAC rules defined earlier are in a package imported as rbac_db.

```
default allow := false

allow = true {
    # Use the `allow` function from `authnz.envopa` to check our RBAC
    # rules against the HTTP request received from Envoy. The function
    # returns the user extracted from the JWT if the check is successful.
    user := envopa.allow(rbac_db)

    # Put below this line any service-specific checks on e.g. the
    # HTTP request received from Envoy.
}
```

Figure 15. Rego policy for importing authnz library

Funded by the European Union

As already mentioned, authnz automatically handles the evaluation of RBAC rules, user authentication, JWT validation, OIDC discovery as well as cryptographic keys download, verification and caching. Also of note, authnz provides built-in functions to evaluate user defined RBAC rules interactively in the Rego REPL. This is useful for dry-run scenarios where a policy writer may want to see what is the effect of their RBAC rules before deploying them to the data lake.

3.3.5 Alternative policy decision points

The Envoy External Authorization Filter can be connected to any gRPC service implementing the External Authorization interface. Thus, policy decision points other than the OPA Envoy Plug-in may be wired into the TEADAL data lake too. Two alternatives were developed as prototypes for this purpose.

One alternative policy decision point is the TEADAL Datalog interpreter. This is a gRPC service that implements the Check procedure by spawning a separate process to interpret Datalog policies. The process runs the Soufflé binary to evaluate a Datalog script against the HTTP request data extracted from the input CheckRequest structure. The service then returns either an "allow" or "deny" response to the Envoy External Authorization Filter depending on whether or not the script allowed the request. Note how the Datalog interpreter's architecture is very similar to that of the OPA Envoy Plug-in. Conceptually the workflow is the same, although instead of evaluating Rego code, the service evaluates Datalog scripts.

Another variation on the theme is Anubis. This is a Web Access Control (WAC) solution originally developed as part of the FIWARE platform, but currently being extended to work in the TEADAL data lake too. The basic idea here is that a product owner uses the Anubis UI to specify access control rules in a WAC-compliant fashion without needing to understand a programming language whereas a programmer develops FDP-specific Rego scripts to evaluate WAC policies against data product HTTP requests. The policy decision point in this case is still the OPA Envoy Plug-in which is preconfigured with the Rego scripts whereas the Anubis policy management service pushes WAC policies to the plug-in at regular intervals.

3.3.6 Access Control Components

This section summarizes the software components providing the access control functionality which has been discussed earlier.

Istio: Message interception facility. Istio's Envoy proxies intercept each consumer request and data product service response.
External Authorization Filter: 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 programming 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 OIDC-compliant Identity Management service.

Funded by the European Union

Keycloak: OIDC-compliant Identity Management service. Consumer services act on behalf of users who have proved their identity through Keycloak.
TEADAL Datalog Interpreter: Alternative policy decision point. It implements a workflow which is conceptually the same as that of the OPA Envoy Plug-in, although instead of evaluating Rego code, the service evaluates Datalog scripts.
Anubis: Alternative policy decision point. Allows data providers to specify access control rules in a Web Access Control-compliant fashion, provided a programmer develops FDP-specific Rego scripts to evaluate policies against data product HTTP requests.

3.3.7 Implementation status

At the time of writing, the message interception, access control delegation, OPA decision point and RBAC framework are already in a useable state, albeit they still need refinement. The Nginx policy store has not been implemented yet. Instead, the CI/CD pipeline packs the Rego code into a Kubernetes secret and mounts it directly on the OPA Envoy Plug-in pod.

As for the alternative decision points the situation is as follows. A Datalog interpreter proof of concept is available, but it has not yet been validated and integrated into the TEADAL data lake. It also lacks the ability to download, verify, cache and update Datalog scripts. The Anubis UI and policy management service still require a substantial amount of work to be fully integrated into the TEADAL mesh and no FDP Rego scripts have been developed yet.

Funded by
the European Union

4. TEADAL PIPELINES

In this section we introduce TEADAL pipelines. A pipeline solution that describes a flow of data from a given source to a destination and a sequence of operations which should be performed on the data. Operations can be transformations, analytics or any type of processing on the data. In the first iteration the TEADAL pipeline is used to populate the data content of a SFDP from a FDP.

Data pipelines that integrate large volumes of data that are geographically distributed face issues of data gravity and friction. Data gravity arises from the fact that large data sets are distributed across geographic locations connected via wide-area networks and the need to minimize resource usage to reduce cost and improve sustainability. Friction is caused by the need to comply with data governance requirements, enforcement of consumer agreements, and by resource constraints. Therefore, data pipelines in such environments provide us with opportunities to optimize their deployment and execution in ways that help mitigate data gravity and friction.

One key feature of TEADAL pipelines is that a pipeline can be a compound pipeline, composed of a set of sub-pipelines, each can run in a different location, all without the need of the FDP pipeline developer[26] to handle the fact that sub-component may run in different locations in the continuum. Thus, enabling opportunities for optimizing data processing and transfer in TEADAL stretched data lake. The pipeline can be partitioned to address data friction or data gravity concerns.

Efficiently implementing data pipelines in a geographically distributed, federated data environment requires a robust architecture that addresses the challenges of data gravity, friction, and the imperative to optimize resource usage.

In the next sections we detail the pipeline architecture, the optimization mechanism that can be applied on the pipeline and the components of the TEADAL pipeline. The optimizations leverage information obtains from the TEADAL data lake monitoring and analysis which is described in Section 2.2

4.1 PIPELINE ARCHITECTURE

For our architecture, we choose the utilization of Kubeflow[27] pipelines, a powerful framework designed for the orchestration and deployment of data workflows. Kubeflow pipelines provide a declarative way to define, deploy, and manage end-to-end workflows. Our choice of Kubeflow is based on its versatility, allowing us to execute a wide range of tasks. Moreover, the flexibility to analyze, partition, enrich, and augment (or inject) new tasks as needed into the pipeline aligns perfectly with the dynamic nature of our federated data architecture.

[26] The developer is responsible for the creating pipelines. See details in deliverable 2.2 - pilot cases' intermediate description and initial architecture of the platform.
[27] https://www.kubeflow.org/docs/components/pipelines/v2/introduction/

Funded by the European Union

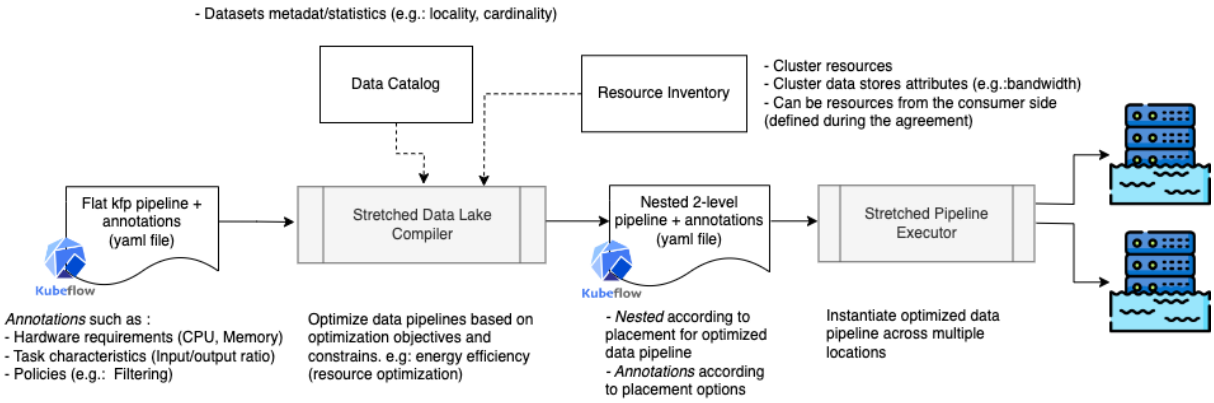

Figure 16. Pipeline optimization architecture

At the core of our data pipeline architecture (Figure 16) is the SDLC, a component designed to analyze, optimize, and enrich data pipelines. Its primary function is to ingest Kubeflow pipelines and apply optimization techniques when deploying the pipeline that will improve the efficiency, resource utilization, and overall performance of data pipelines. This component addresses key optimization objectives, with a primary focus on resource optimization, and other crucial constraints. In doing so, it leverages various data and resource inventory services, including the data catalog, and the Kubestellar Resource Inventory.

On the one hand, the SDLC integrates with the Federated Data Catalog service, offering it a general view of the available Federated Data Products within the federated data lake. This integration, in turn, equips the compiler with valuable information and metadata about the source Federated Data Products used in the Kubeflow pipelines. In addition, the compiler is connected to the data catalog, which provides a detailed understanding of the Federated Data Products, including characteristics such as locality, cardinality, and more.

On the other hand, in order to optimize data pipelines effectively, the SDLC also integrates with the Kubestellar Resource Inventory. This service maintains a detailed inventory of available locations (i.e., clusters) for pipeline execution. These clusters encompass both local clusters within the federated data lake, and consumer clusters located as part of the data consumer infrastructure that can be employed for pipeline execution when necessary. Through this integration, the compiler gains access to crucial information about these clusters, including their resource availability, and data store attributes like bandwidth.

Following the analysis of the input pipelines, the compiler proceeds to create logical groupings of tasks, known as sub-pipelines, based on the insights given by the different services. Leveraging the metadata provided by the data catalog and inventory services, the compiler can make informed decisions tailored to the unique characteristics and requirements of each dataset. With the insights provided by the Kubestellar Resource Inventory, the compiler can intelligently annotate sub-pipelines with information about potential target clusters, guaranteeing efficient resource utilization and minimizing data transfer bottlenecks. In this sense, these sub-pipelines are created based on shared characteristics and requirements of each individual task of the original pipeline, enabling them to be executed within the same cluster. Furthermore, the compiler annotates these sub-pipelines with the optimization decisions it has made. As a result, the output of the compiler consists of an enriched Kubeflow pipeline YAML file, ready for ingestion by the Stretched Pipeline Executor.

The optimized pipelines, enriched with annotated sub-pipelines, are then handed over to the Stretched Pipeline Executor. The Executor receives the enriched pipelines containing annotated sub-pipelines, indicating the most suitable clusters for their execution. Leveraging

Funded by the European Union

the insights and recommendations provided by the SDLC, the executor takes charge of selecting the optimal clusters in the continuum for each sub-pipeline's execution. This decision-making process is based on careful evaluation of factors like resource availability, data store attributes, and overall efficiency.

Note that the clusters in the TEADAL architecture are divided between a control cluster and remote clusters where sub-pipelines can be executed. This division aligns seamlessly with the executor's role in selecting optimal clusters for sub-pipeline execution. The control cluster comprises the Stretched Pipeline Executor and the Kubestellar controller, which is aware of all the remote clusters. In each remote cluster, there is an instance of Kubeflow pipelines and a Stretched Pipeline Executor "agent" functioning as a Kubernetes controller. This agent is responsible for receiving sub-pipelines from the control cluster and executing them within the local Kubeflow instance.

In this sense, the output of the Stretched Pipeline Executor, a custom Kubernetes object, is applied locally in the control cluster within Kubernetes, housing the Kubestellar controller. Kubestellar then transparently dispatches the Kubernetes object to the target remote cluster, where the Stretched Pipeline Executor "agent" receives and executes the sub-pipelines in the local Kubeflow instance.

One of the key responsibilities of the executor is orchestrating the connections between different sub-pipelines, especially when they are set to run in separate clusters. These connections are vital, as the output of one sub-pipeline serves as the input to another. The executor's logic ensures that data flows smoothly between sub-pipelines, regardless of their physical location. In this sense, the executor minimizes data transfer bottlenecks and simplifies the overall pipeline execution process by making intelligent decisions on data transfer and inter sub-pipeline connectivity.

4.2 OPTIMIZING DATA PIPELINES

The grouping and placement of data processing tasks across a geo-distributed environment (i.e., the cloud continuum) has an impact on the overall characteristics of a data pipeline. For example, depending on the co-placement of tasks across multiple processing locations, it is possible to reduce the overall energy consumption, minimize the amount of data that needs to be transferred among tasks across the wide-area network (e.g., edge to cloud), and reduce cost or execution time. This needs to be done while taking into account various constraints, such as data governance, and hardware requirements. We want to determine grouping and placement in an automated manner to relieve the FDP pipeline developer of having to reason about the inherent characteristics and constraints which are defined by the FDP designer[28].

In the first instance, we focus on placement of tasks across provider-controlled sites and consumer-controlled sites. We develop an initial demonstration of the capabilities to group data pipeline tasks across these locations in a manner that is compliant with governance contracts while balancing execution across both locations to achieve some optimization/minimization objectives. More specifically, we want to demonstrate the placement of governance operations that should avoid passing of certain information between sites at the provider-controlled site, while placing other data transformation or formatting tasks in a manner that minimizes the amount of data that needs to be transferred across sites.

Eventually it may be possible to realize multi-objective optimization that achieves near-optimal tradeoffs of the various concerns imposed by gravity and friction without putting the onus for

[28] The designer is responsible for creating policies and metadata. See details in deliverable 2.2 - pilot cases' intermediate description and initial architecture of the platform.

Funded by the European Union

the required decision-making on the FDP pipeline developer. Examples of actions that can be taken to optimize data pipelines are reordering of tasks, injection of tasks, and grouping of tasks by location.

4.2.1 Leveraging metadata for optimizations

Optimizations applied to data pipelines are based on metadata about datasets, operations, and the compute environment. Metadata about datasets (from the data catalog) can include for example, the size of the dataset and the distribution of data. Metadata about operations or task attributes include the input-output ratio of tasks, information about the columns being read and/or written, the nature of updates to the columns being written (e.g., append-only, overwrite). An optimizer for data pipelines can also make use of information that describes the characteristics of the compute environment, such as bandwidth between locations, the availability of hardware resources, and the geographic location of compute clusters. These different types of metadata support a data pipeline optimizer with the attainment of optimization objectives (e.g., minimize data transfer, energy consumption, cost, execution time) at the same time as it takes constraints in the form of governance policies and task hardware requirements into account.

There are multiple sources of relevant metadata. Statistics about datasets are typically collected in technical metadata stores. Task characteristics can be provided by the users specifying a data pipeline in the form of task annotations or they can be derived automatically (e.g., with the use of machine learning). Information about the compute environment is maintained in resource inventories.

In the first instance, we use metadata that allows us to optimize the placement of tasks to minimize the transfer of data across sites while taking simple governance contracts into account. More specifically, we use the size of input datasets along with the input-output ratio of tasks for placing tasks in a way that minimizes data transfer across wide-area networks or across cloud providers. For the first iteration, we assume that statistics about datasets are available as input to the optimizer (Stretched Data Lake Compiler). Similarly, we assume that users annotate tasks with their input-output ratios. In subsequent operations we aim to replace these manual sources with data catalogs and more automated approaches. In addition, we demonstrate the capability of placing tasks in a way that complies with geographic / site constraints.

4.2.2 Exploring predicting metadata in lieu of collecting

In addition to automating the pipeline optimizations, it is also desirable to relieve the FDP pipeline developer from having to specify some of the task characteristics that are used by the data pipeline optimizer. Putting the onus for this on the FDP pipeline developer may result in inaccurate estimates and/or require additional effort to take measurements. Instead, we want to produce accurate estimates of relevant metadata, such as task characteristics in an automated manner. For this purpose, we have begun an exploratory course of work to investigate how we can accurately predict task characteristics (e.g., output cardinality or output size of a data processing task) given some relevant metadata (e.g., input dataset size, input data types, input data distributions) and a specification of a task in a data pipeline, such as SQL query text or the source code of a user-defined function (UDF).

The database optimization community has been working on related questions. The state-of-the-art [1, 2, 3, 4] for estimating the cardinality of SQL queries consists of machine-learning based approaches. However, viewed in our context, existing work suffers from an important shortcoming. The relevant features of a task (e.g., SQL query, UDF) need to be determined

Funded by the European Union

manually. In addition, models may need to be retrained per group or type of SQL query or even for each new UDF. This does not scale well to the case of data pipelines that typically consist of a mix of tasks that are representable as SQL operations and ones that are not so readily expressed through SQL. Data pipelines make use of UDFs in order to achieve domain adaptability and to express complex logic more conveniently than would be possible with SQL. Therefore, in our case, we want to avoid the need for manual featurization as much as possible.

We investigate whether the ability of Large Language Models (LLM) to discover complex statistical patterns within language and program source code results in embeddings of SQL query text and of UDF source code that represent relevant features for the accurate estimation of output metadata (e.g., output cardinality or size). We train an LLM encoder, which has been previously trained on programming language text (e.g,. Python, SQL), together with a regression head (e.g., a simple neural network) on samples of task source code and output cardinality for a given database and evaluate its accuracy on unseen tasks. Initial experiments that compare our approach with one SoTA approach [1] for SQL query cardinality estimation indicate that our approach can achieve comparable accuracy, even before applying any fine-tuning, hyper-parameter optimization, or use of more sophisticated regression heads.

This work is entirely exploratory in nature. There is a large space of ideas we are going to explore experimentally to find out if we can confirm and improve on our initial results. Ideally, we would be able to propose an approach to accurately predict various types of task characteristics as they are needed for data pipeline optimization in an automated manner and avoiding the need for manual feature identification.

4.2.3 Optimization mechanisms

Optimization of database queries has become commonplace with various rule-based and cost-based optimizers for SQL queries. It would be beneficial to be able to apply relevant optimizations to data pipelines that integrate large volumes of data that are spread across locations so that we can run these data pipelines as sustainably as possible. Co-location of tasks to minimize data transfer, reordering of tasks to move processing closer to the source, injection of tasks to increase efficiency of operations (e.g., compress data prior to WAN transfer or pre-warm a cache at a remote location) are examples of data pipeline optimizations based on metadata. It is also conceivable to inject tasks in order to comply with governance policies (e.g., remove a column or mask its contents).

Many of these optimizations require innovation. In order to enable valid and useful reordering of data pipeline tasks that consist of a mix of relational and non-relational operators, we will need to develop semantic frameworks that can be used to represent relevant information about tasks. We need to learn what information is indeed necessary to support automated reordering of arbitrary logic as it exists in data pipelines in the form of UDFs, how much of it can be obtained in an automated manner, and how to facilitate integration of such information for new types of tasks by the FDP pipeline developer. To support task grouping and placement decisions, we will need to investigate multi-objective optimization and constraint satisfaction to reason about multiple optimization objectives (e.g., data transfer volume, cost, execution time) while taking constraints into account (e.g., budget, data governance). In addition, we may want to explore learning-based approaches to predict the resource requirements of data pipeline tasks.

In the first instance we will focus on grouping tasks in a data pipeline across locations in a way that minimizes data transfer across the network while complying with location constraints for tasks. We chose to follow a constraint programming approach [5] for the current implementation, but may investigate the use of graph partitioning algorithms for future iterations. Given a data pipeline with annotations specifying task hardware requirements and

Funded by the European Union

location constraints, a Kubestellar Resource Inventory providing information about sites, we find assignments of groups of tasks that minimize cross-site data transfer and complies with specified constraints. First, we determine feasible sites based on matching the resource inventory with task requirements and location constraints. This step produces the set of valid sties per data pipeline tasks. Next, we apply constraint optimization to find all cuts that satisfy the valid sites per task while minimizing an objective function (i.e. cross-site data transfer volume). The result is an assignment of groups of tasks to sites that complies with unavoidable cuts (e.g., due to hard constraints such as hardware requirements and governance) while minimizing some notion of cost. We repeat the execution of the constraint solver to provide a ranked list of solutions.

Optimizations are performed by the Stretched Data Lake Compiler whose inputs consist of:

1. A data pipeline specification in the form of a Kubeflow pipeline that is annotated with task characteristics (e.g., input-output ratio, location constraints, hardware requirements), references to datasets in a way that affords looking up dataset statistics.
2. In this iteration, part of the user-provided annotations are location constraints for tasks that allow the Stretched Data Lake Container to determine permissible sites per task, hardware requirements and task input-output ratios. In later iterations, we will aim to augment this with information about permissible locations per dataset that can be processed to automatically determine permissible placement of tasks operating on a given dataset.
3. Metadata about datasets (provided manually in first iterations and to be replaced with programmatic lookup in metadata catalog in later iterations).
4. Kubestellar Resource Inventory information that lists each site / cluster, information about its geographic location, and its resources.

The data pipeline specification and other inputs are combined into an Intermediate Representation (IR) that the internal components of the SDLC know to operate on. A constraint satisfaction solver translates constraints and hardware requirements into a list of allowed clusters per task. The set of clusters for each task are chosen so as to satisfy the governance constraints and hardware requirements. This in turn serves as input to a component that computes the data output size of each task to group tasks in a way that minimizes cross-cluster data transfer within the constraints of valid clusters per task. Finally, the SDLC generates a set of nested data pipeline specifications (i.e. nested Kubeflow Pipelines). Each element in this set represents one solution ranked from the best one (i.e. minimal amount of cross-location/site data transfer) to less optimal ones. This provides the Stretched Data Lake Executor with some degrees of freedom to choose deployments according to current conditions in the compute environment. Each sub-pipeline within a solution represents co-location of a group of tasks. The nested pipelines serve as input to the Stretched Data Lake Executor.

4.3 DATA PIPELINE COMPONENTS

For implementing the multi-location data pipelines, we have developed the following components.

4.3.1 Stretched Data Lake Compiler

The Stretched Data Lake Compiler is the core of our system, handling the complexities of optimizing data pipelines in a federated environment. It thoroughly analyzes and enhances Kubeflow pipelines, connecting with crucial services like the data catalog, and Kubestellar Resource Inventory. The compiler integrates with the Federated Data Catalog for a broad view of the data products in the data lake. Simultaneously, it connects with the data catalog for a

Funded by the European Union

detailed understanding of data product features. This collaborative approach extends to the Kubestellar Resource Inventory for a detailed view of the available resources, ensuring informed decisions for creating sub-pipelines tailored to each dataset. The result is a Kubeflow pipeline enriched with optimization decisions for efficient execution by the Stretched Pipeline Executor.

Note that the implementation of the SDLC is based on a component called Multicloud Computer Compiler (MCC-C). The current implementation of the SDLC is based on IBM's work on MCC-C, which aims to address the more general case of pipelines operating across multiple clouds. While we contribute a version of MCC-C that caters to the requirements of TEADAL, we would like to maintain parity with the MCC-C codebase as much as possible to be able to make it available for a variety of additional use cases.

4.3.2 Stretched Pipeline Executor

The Stretched Pipeline Executor is the main component in executing our optimized data pipelines. This component selects optimal clusters for executing annotated sub-pipelines thanks to the insights given from the compiler. Its decision-making process is based on evaluating resource availability, data store attributes, and overall efficiency. The Executor not only executes but also coordinates connections between sub-pipelines, moving data through local or remote clusters. This coordination is crucial for ensuring the appropriate data pipeline between sub-pipelines, regardless of their location. By intelligently managing data transfer and inter sub-pipeline connectivity, the Executor minimizes issues and simplifies the entire execution process. Essentially, it acts as the orchestrator of the pipeline.
The executor is using the Kubernetes API to execute the sub-pipelines and is taking advantage of Kubestellar to deploy the pipeline into the TEADAL data lake, which is realized by one or more Kubernetes clusters.

Funded by the European Union

5. TOWARDS NEXT ITERATION

In this document we summarized the initial iteration of the TEADAL data lake which is focused on defining and developing the building blocks of the data lake.

In the second iteration we aim to:

1. Extend the data lake monitoring and add analysis and insight capabilities, creating metadata that will help optimize the TEADAL pipeline, and the overall use of TEADAL data lake.
2. Extend into multiple locations and demonstrate how the control plane simplifies management of a multi-location data lake.
3. Demonstrate how a pipeline is partitioned for the purpose of implementing energy efficiency, data gravity, data friction policies, and to be able to apply the insight developed from monitoring and observing the data lake.

The exploration and developed described above would be validated and demonstrated during the second iteration with the pilot cases.

Funded by
the European Union

REFERENCES

[1] Kipf et al., (2019). Learned Cardinalities: Estimating Correlated Joins with Deep Learning, Conference on Innovative Data Systems (CIDR), 2019

[2] Boulos, J., & Ono, K. (1999). Cost Estimation of User-Defined Methods in Object-Relational Database Systems. *SIGMOD Rec.*, *28*(3), 22–28. https://doi.org/10.1145/333607.333610

[3] Dutt, A., Wang, C., Nazi, A., Kandula, S., Narasayya, V., & Chaudhuri, S. (2019). Selectivity Estimation for Range Predicates Using Lightweight Models. *Proc. VLDB Endow.*, *12*(9), 1044–1057. https://doi.org/10.14778/3329772.3329780

[4] Boulos, J., Viemont, Y., & Ono, K. (1997). A neural networks approach for query cost evaluation. *Transaction of Information Processing Society of Japan*, *38*(12), 2566–2575.

[5] Google OR-Tools CP-SAT solver, https://developers.google.com/optimization/cp

Funded by the European Union