Deliverable D2.1 – Architecture design and component definitions

GRANT AGREEMENT NUMBER: 101070141

This project has received funding from the European Union's HE research and innovation programme under grant agreement No 101070141

GLACIATION

Project acronym:	**GLACIATION**
Project full title:	**Green responsibLe privACy preservIng dAta operations**
Call identifier:	**HORIZON-CL4-2021-DATA-01-01**
Type of action:	**RIA**
Start date:	**01/10/2022**
End date:	**30/09/2025**
Grant agreement no:	**101070141**

D2.1 – Architecture design and component definitions

Executive Summary:	D2.1 provides the GLACIATION platform architecture blueprint together with the service components definition		
WP:	2		
Author(s):	Flavio Scaccia, Rosario Catelli		
Editor:	Rosario Catelli		
Leading Partner:	ENG		
Participating Partners:	EISI, HIRO, LUH, UNIMI, UNIBG, EURECOM, SAP SE, UCC, SOGEI, LAKE		
Version:	1.0	**Status:**	Submitted
Deliverable Type:	R - Document, report	**Dissemination Level:**	PU - Public
Official Submission Date:	31/03/2024	**Actual Submission Date:**	31/03/2024

Disclaimer

The GLACIATION consortium consists of the following partners:

No.	Partner Organisation Name	Partner Organisation Short Name	Country
1	MINISTERO DELL'ECONOMIA E DELLE FINANZE	MEF	IT
2	EMC INFORMATION SYSTEMS INTERNATIONAL UNLIMITED COMPANY	EISI	IE
3	HIRO MICRODATACENTERS B.V.	HIRO	NL
4	GOTTFRIED WILHELM LEIBNIZ UNIVERSITAET HANNOVER	LUH	DE
5	THE LISBON COUNCIL FOR ECONOMIC COMPETITIVENESS ASBL	LC	BE
6	UNIVERSITA DEGLI STUDI DI MILANO	UNIMI	IT
7	UNIVERSITA DEGLI STUDI DI BERGAMO	UNIBG	IT
8	GEIE ERCIM	ERCIM	FR
9	EURECOM	EURECOM	FR
10	SAP SE	SAP SE	DE
11	UNIVERSITY COLLEGE CORK - NATIONAL UNIVERSITY OF IRELAND, CORK	UCC	IE
12	SOGEI-SOCIETA GENERALE D'INFORMATICA SPA	SOGEI	IT
13	LAKESIDE LABS GMBH	LAKE	AT
14	ENGINEERING - INGEGNERIA INFORMATICA SPA	ENG	IT
15	EIDGENOESSISCHE TECHNISCHE HOCHSCHULE ZUERICH	ETH	CH

Document Revision History

Version	Description	Contributions
0.1	Table of Content	ENG
0.5	1st version	All
0.7	1st reviewed version	HIRO, ETH
0.9	2nd reviewed version	EISI
1.0	Final Version	MEF, ENG

Authors

Authors	Partner
Flavio Scaccia, Rosario Catelli	ENG
Liam O'Toole	UCC
Marco Abbadini, Michele Beretta, Dario Facchinetti, Stefano Paraboschi, Matthew Rossi	UNIBG
Sabrina De Capitani di Vimercati, Sara Foresti, Pierangela Samarati	UNIMI
Oleksandr Chepizhko, Péter Forgács	LAKE
Enrico Chiacchiari	SOGEI

Reviewers

Author	Partner
Konstantin Tatarnikov	HIRO
Maciej Besta	ETH

Table of Contents

List of Figures

List of Terms and Abbreviations

Abbreviation	Description
AI	Artificial Intelligence
API	Application Programming Interface
BRAINE	Big data pRocessing and Artificial Intelligence at the Network Edge
DFD	Data Flow Diagram
DKG	Distributed Knowledge Graph
iDRAC	integrated Dell Remote Access Controller
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
ISO	International Organization for Standardization
K8s	Kubernetes
ML	Machine Learning
MOSAICrOWN	Multi-Owner data Sharing for Analytics and Integration respecting Confidentiality and OWNer control
MPC	Multi-Party Computation
PaaS	Platform as a Service
RDF	Resource Description Framework
REST	REpresentational State Transfer
SLA	Service Level Agreement
SPARQL	SPARQL Protocol and RDF Query Language
UI/UX	User Interface / User Experience

Executive Summary

The GLACIATION project, a pioneering European research initiative, has achieved a significant milestone with the completion of Deliverable D2.1, focused on the development of the GLACIATION platform architecture blueprint and the definition of its service components. This deliverable marks a crucial step towards realizing the project's overarching objectives of creating a versatile and adaptable framework that interfaces seamlessly with both Cloud and Edge microservices.

The primary objective was to design and develop a modular and pluggable architecture that aligns closely with the diverse requirements of the project's use-cases. The architectural blueprint of the GLACIATION platform was crafted with meticulous attention to detail, considering all pertinent use-case requirements. Privacy, security, scalability, and modularity were identified as critical factors, ensuring that the resulting blueprint is robust and flexible. Thorough evaluations were conducted to assess the impact of technology stack choices, cots solutions, and partner tools on the architecture.

Each service component within the architecture underwent rigorous scrutiny to define its functionality, interfaces, and essential requirements. Whether realized through cots solutions, existing partner tools, or subsequent work-packages, every component was meticulously documented to facilitate seamless integration and future scalability.

In conclusion, Deliverable D2.1 represents a significant leap forward in the GLACIATION project, providing a robust architectural blueprint and comprehensive service component definitions. Moving forward, this foundational work will serve as a guiding framework for the subsequent phases of development, ensuring the successful realization of the project's ambitious goals.

1. Introduction

This document covers the software architecture of the GLACIATION project, which aims to provide a modular platform for secure and efficient data processing, management, and workload placement. As well as an overview and description of the components included in the system.

1.1. Objectives of the deliverable

The scope of the document includes the different views of the architecture, as well as decisions and technologies used in the project. The document will also cover the integration of GLACIATION with other European projects such as BRAINE and MOSAICrOWN and with advanced open-source tools.

The document does not cover the hardware architecture, or the specific algorithms used for data processing, but rather to provide an overview of the GLACIATION software architecture. It describes the high-level architecture of the system and provides more detailed diagrams/viewpoints of the various components and their interactions. The scope of this document is to provide a comprehensive understanding of the software architecture and its functionality for multiple stakeholders. Later in the document, the focus will shift towards delving into the implementation details, starting from the high-level architecture, and gradually moving towards more specific and detailed diagrams and viewpoints. It will serve as a reference for software architects, developers, and stakeholders involved in the design and development of the system.

This architecture description is based on a thorough analysis of the relevant state of the art and is guided by the initial requirements extracted from the pilots and, of course, by the GLACIATION vision and goals.

The GLACIATION project primary goal is to develop an architecture that enables efficient, secure, and scalable data movement across heterogeneous environments, enabling the edge to cloud continuum, while optimizing resource utilization. One of the key influences on the architecture decisions is the emphasis on modularity. Modularity is considered an essential aspect of the system as it allows for flexibility, extensibility, and easy integration of new components. The architecture is designed to be composed of modular building blocks that can be independently developed, deployed, and updated. This modular approach enables the system to adapt to evolving requirements, accommodate diverse use cases, and facilitate future enhancements.

By adopting a modular architecture, the GLACIATION project promotes component reusability, which leads to reduced development efforts and improved system maintainability. It allows different modules to be developed by different teams or organizations, fostering collaboration, and fostering innovation in the ecosystem.

Furthermore, modularity enables the system to scale horizontally by adding or removing modules based on demand. This scalability ensures that the architecture can handle varying workloads and accommodate the dynamic nature of distributed systems. The modularity

aspect also contributes to the system's resilience and fault tolerance. Since components are decoupled and can operate independently, failures or issues in one module do not affect the overall system's stability. Faulty components can be isolated and replaced without disrupting the entire system, resulting in increased reliability and availability.

1.2. Relation with other tasks and deliverables

This deliverable is the result of Task 2.1 (Design of the architectural blueprint of the GLACIATION platform) and Task 2.2 (Component Definitions) – where all the requirements of the use-cases were listed and considered, supporting the architecture design based on functional requirements. D2.1 will be useful as a basis for the technical implementation, validation, and demonstration of the overall GLACIATION platform. Other technical tasks include establishing communication between components and transforming the architecture into a functional and usable system.

1.3. Document Structure

The remaining part of the document is structured as follows:

- Chapter 2 provides an in-depth exploration of the high-level components constituting the GLACIATION architecture. The chapter introduces the methodology employed in crafting the architecture and employs various diagrams to offer a comprehensive understanding from different perspectives.
- Chapter 3 delves into the core of the system – the AI Movement Engine. It offers a detailed description of the orchestrator, clarify the decision-making process fuelled by collaborative artificial intelligence methods, swarm techniques, and insights derived from the Distributed Knowledge Graph.
- Chapter 4 delves into the critical aspect of security within the GLACIATION system. It explores the mechanisms adopted to ensure the robust safeguarding of the system's components.
- Chapter 5 focuses on the Performance Measurement Framework, this chapter illustrates how energy considerations are integrated into the framework, contributing to an understanding of the system's performance.
- Chapter 6 introduces a set of services specifically designed for comprehensive data management, from data storage to metadata services, replica management, trade-offs, data processing monitoring, and prediction services, this chapter covers a spectrum of functionalities to handle diverse data-related needs.
- Chapter 7 includes observability services for monitoring and logging of components contributing to the overall maintenance of the system.
- Chapter 8 is dedicated to the critical management services essential for controlling and creating the GLACIATION environment.

2. Architectural Representation

To ensure a clear understanding of the system, it is important to use different diagrams and viewpoints that can better illustrate the system's architecture, design, and behaviour. For this reason, this section describes different Views: Use-case View, Logical View, Deployment View and finally a deeper technical view with the DFD.

2.1. Architecture design methodology

We developed our architectural framework in accordance with the widely used ISO/IEC/IEEE 42010:2022[1]. Through the formation of architecture descriptions, this standard is essential in guiding the construction, analysis, and sustainability of system architectures. The standard defines the key components that must be included in architectural descriptions and provides a conceptual paradigm for doing so. Furthermore, it emphasizes the importance of incorporating various architecture viewpoints to ensure a comprehensive understanding of the system. For this reason, follows a description of different viewpoints and associated diagrams.

2.2. Use-case View

The GLACIATION project aims to enhance the efficiency and effectiveness of reliable digital technologies by addressing the needs of citizens, companies, and public organizations. These requirements span privacy, commercial confidentiality, administrative confidentiality, and environmentally friendly data operations across the entire data life cycle. GLACIATION will showcase its solution through three real-life scenarios: a national government-wide platform, an Industry 4.0 setting, and cross-company collaborations. These scenarios will benefit from optimized data movement and reduced power consumption.

The architectural design of GLACIATION aligns with the specific requirements of three use cases, encompassing public service, manufacturing, and enterprise data analytics. Each use case is elaborated upon in the following sections.

2.2.1. UC1 View

MEF's use case will be based on the NoiPA platform to improve data movement and service delivery to a local (decentralized) level through edge technology. MEF would also like to move towards AI/ML engine adoption where possible.

The actual system is composed by different component:

- Remote Site
    - Turnstiles: this is the system responsible to write Turnstile stamps

[1] https://www.iso.org/standard/74393.html

- Concentrators: this system collects raw data from all turnstiles in the buildings. There can be more than one building or site per Public Administration, so there can be different concentrators depending on networking connections.

- Central Site:

  Platform based on Redhat openshift, below are the main architectural components:

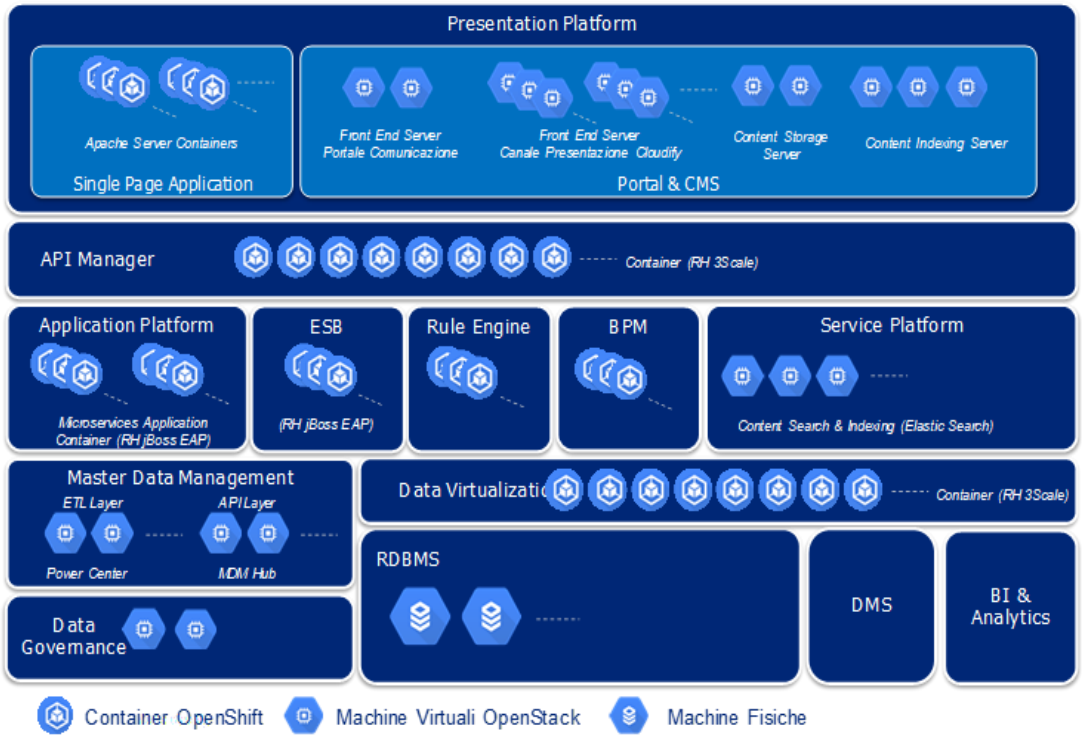

Figure 1: Architectural Components.

The logical architecture of the web application includes a three-level model Presentation, Business and Data. The layer-based approach allows you to limit coupling and functional duplications, promoting the creation of highly specialized and cohesive objects.

- **User Interface (UI or presentation layer):** has the purpose of managing the system's interaction with the outside world, in particular with users. It includes masks for viewing and entering data, controls, from the simplest to the most complex, and mechanisms for intercepting and appropriately dealing with events that are triggered based on the actions carried out by users.

- **Business Logic Layer (BLL or business logic):** includes the set of business rules that regulate the operation of the application, intercepts requests coming from the presentation layer and manages them appropriately.

- **Data Access Layer (DAL or data access layer):** deals with accessing and persisting the information processed by the application and knows how to read and save it within a data source (not necessarily a relational database). Access to data is mediated by a virtualizer which allows you to integrate relational and non-relational sources with each other.

- **Database:** constitutes the operational database as a data source to support data collection and presentation activities, and for the automation of procedural activities through access, by services, for insertion, modification, reading and aggregation.
- **External Services:** made up of services and libraries for common use of the platform.

The implementation of the architecture in the UC1 context is provided in the figure below.

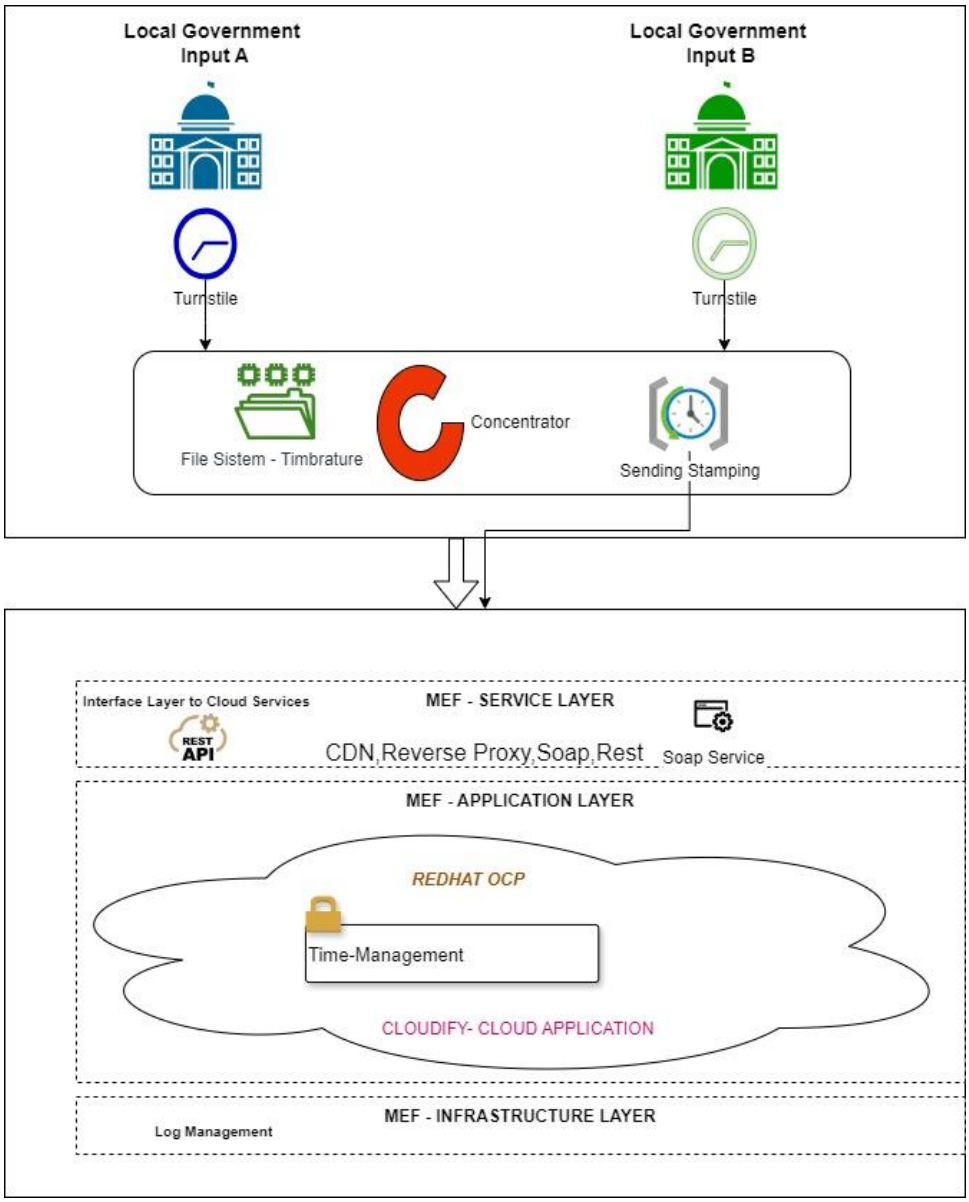

Figure 2: UC1 Overview.

To implement the use case, using the GLACIATION platform, the following architectural transactions are expected:

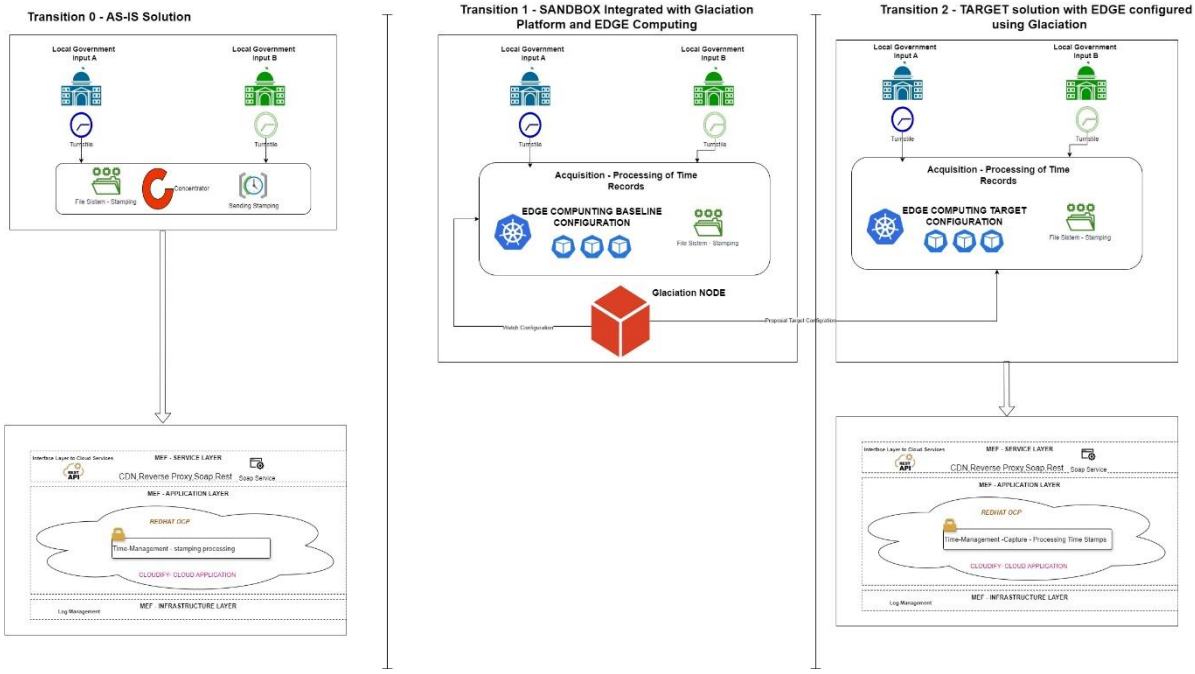

Figure 3: Architectural Transactions.

2.2.2. UC2 View

Dell's use case UC2 is centred at the Dell Manufacturing facility in Cork, Ireland. The pilot involves adapting software that analyses data collected by all the robots within the facility to run on the GLACIATION platform. The GLACIATION platform will ingest this data and anticipate three key outcomes for validation:

1. Energy Reduction in Manufacturing – Achieved by transitioning data analytics to an edge-core-cloud platform, leveraging AI/ML technologies.
2. Enhanced Predictive Analytics – By ingesting data with metadata annotations into the distributed graph of GLACIATION.
3. Optimized Energy Consumption – Achieved through efficient data movement across the manufacturing GLACIATION platform.

The UC2 Validation Framework is as shown in the Figure 4, below:

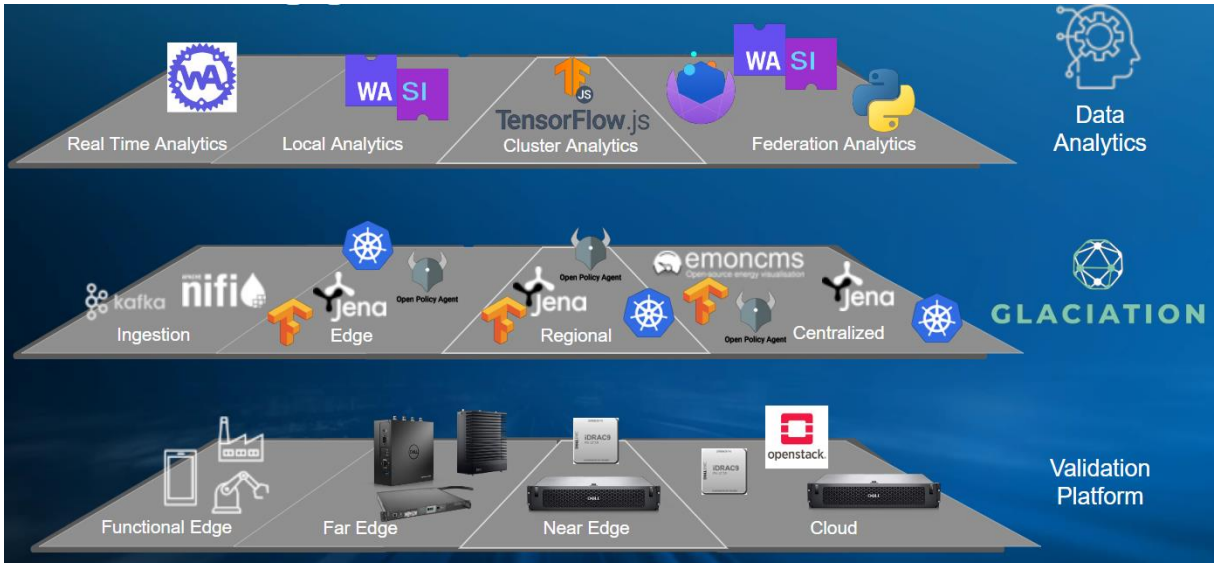

Figure 4: UC2 Validation Framework.

For UC2, we are constructing the platform using Dell Harbour, integrating environments that span from the cloud to the edge. We utilize Dell servers and gateways to create this integration environment. The GLACIATION platform sits atop this infrastructure, managing user data. Our goal is to optimize data analytics on the GLACIATION platform for both energy efficiency and privacy.

The GLACIATION platform will begin ingesting data from the cobots, including diagnostic information about the robots themselves, alongside the functional data specific to each cobot. This approach will lead to reduced energy consumption in manufacturing, improved predictive analytics, and minimized power usage for analytics and predictions.

The manufacturing use case within the project seeks to harness advanced data analytics and energy-efficient practices for optimizing manufacturing processes. As part of Dell's commitment to sustainability and innovation, this use case incorporates cutting-edge technologies to enhance energy efficiency while upholding data privacy and security.

2.2.3. UC3 View

SAP's use case UC3 focuses on fostering data-driven business collaborations by allowing to reveal only what is strictly required, the insights over joint data, and not share entire sensitive business data with multiple business partners. The overall aim is to provide a generic solution to generally foster cross-company collaborations with high security and privacy guarantees. In the context of the use case, we specifically investigate asset performance management. For example, an equipment and machinery manufacturer wants to gather insights from various customers who use their assets – to learn how it is used, what can be improved, and improve prediction accuracy.

However, customers are hesitant to participate because sharing this data could reveal sensitive production details and provide information about a company's current business situation. Thus, our need to protect the data and only reveal the required insights but not the underlying data. To facilitate collaborations while always protecting data, we leverage privacy-

enhancing technologies. Mainly, secure MPC to enable to run computations on encrypted data in a collaborative fashion as suitable for cross-company collaborations.

To simplify acceptance for business customers, such solutions must be compatible with cloud-native deployment. In an enterprise context, this requires support for SAP Business Technology Platform (BTP), a cloud solution designed for SAP applications. BTP is a PaaS provider that serves as a strong foundation for developing and deploying applications that integrate seamlessly with SAP services in the cloud, like S4/HANA.

We leverage different technologies to realize privacy-preserving enterprise collaborations:

- For BPT-suitable cloud-deployment, we leverage Gardner[2].
- For MPC deployment, we leverage Bosch Research's open-source cloud stack for MPC solutions, i.e., Carbyne Stack[3], where SAP is a contributor.
- For MPC execution, MP-SPDZ[4] is the MPC backend for MPC implementations, supported by Carbyne Stack.

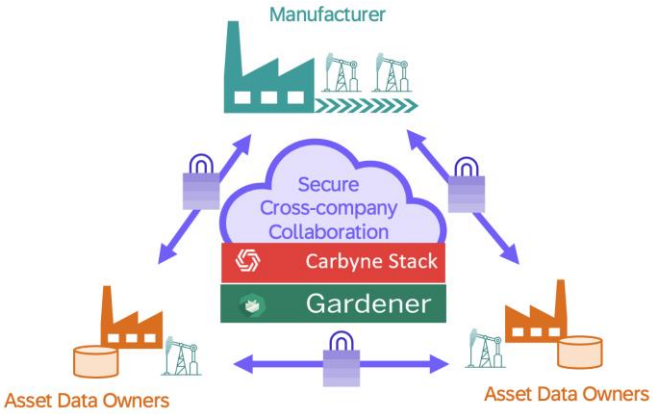

Figure 5: UC3 Overview.

2.3. Logical View

This section, Logical View, describes the high-level software components and their relationships in terms of the functionality they provide. It focuses on the logical organization of the software components and their behaviour, without going into details of the implementation. This view provides a clear understanding of how the GLACIATION system is structured. It also merely shows a high level of superficial dependencies and relationships between the components; the next chapters, which are devoted to the description of each component, describes the deeper interfaces via which they communicate.

[2] https://gardener.cloud/
[3] https://carbynestack.io/
[4] https://github.com/data61/MP-SPDZ

2.3.1. Layers

The multi-layered architecture of the system can be divided into several levels for a better understanding of the overall structure.

At the highest level (L1), there are multiple physical clusters that form the backbone of the system, each responsible for managing a specific set of nodes and services.

Within each cluster, there are multiple nodes (L2) that are responsible for running services and workloads. These nodes can be further divided into smaller logical groupings based on their specific functions and capabilities, being them control or worker nodes.

At the lowest level (L3), there are individual physical nodes that make up the clusters and practically host the various services and workloads. These nodes can be distributed and replicated to ensure redundancy and scalability.

Finally, at the application layer (L4), there are various microservices and containerized applications that provide the business logic and functionality of GLACIATION. These microservices are designed to be modular and independently deployable, allowing for flexibility and agility in the development and deployment of the system. This layer is not represented in the next diagram but is fully described in the next section.

At present, although designed, there is no requirement for multi-site: consequently, the test lab will be set up with one cluster.

Follows Figure 6 representing the described layers and a further description of each.

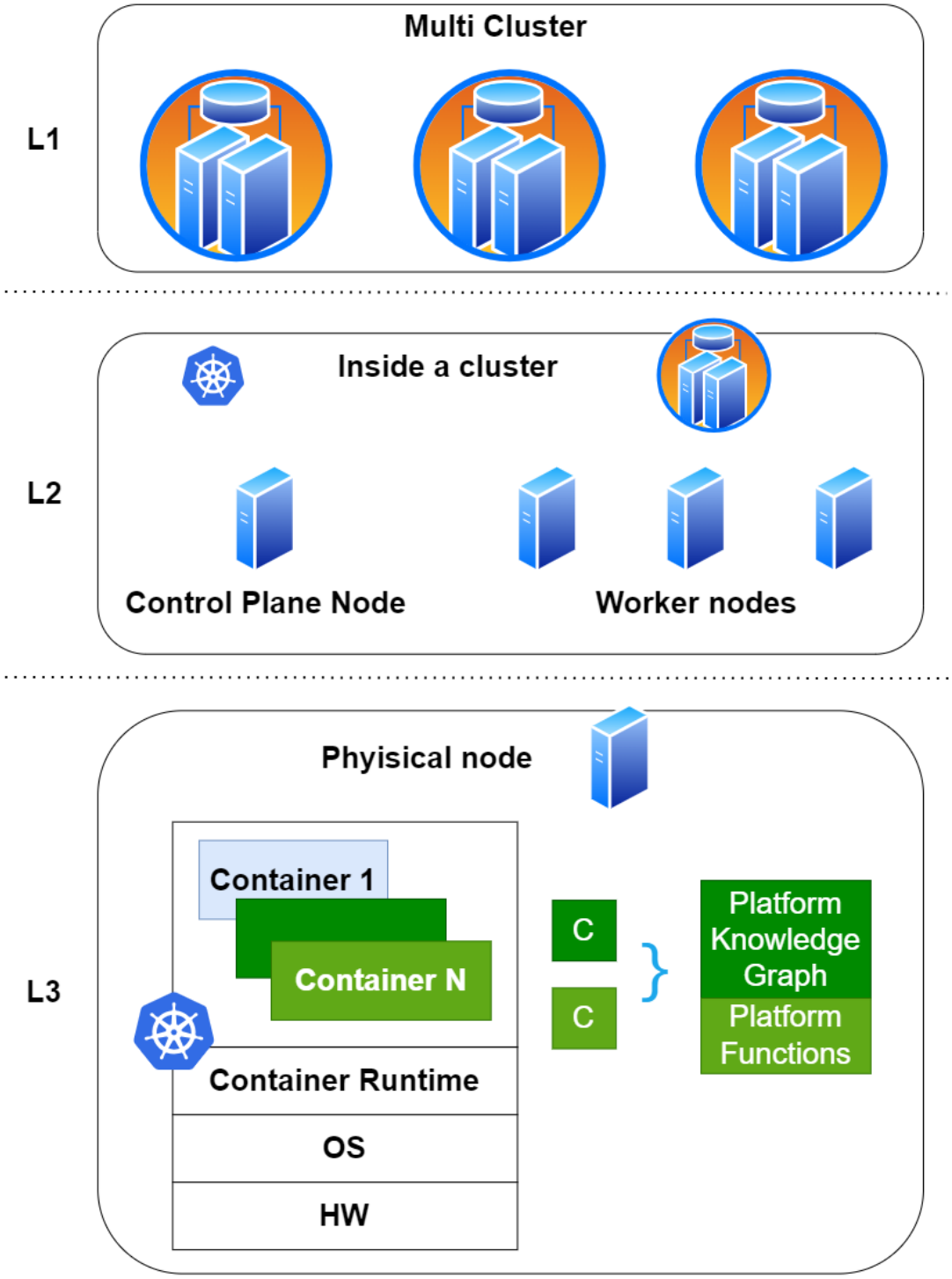

Figure 6: Architecture Multi-Layer View.

Description L1 – Multi Cluster

Level 1 focuses on the organisation and management of multiple clusters to ensure that the clusters are secure, always available, and easy to manage. This is where a user interacts with the multi-cluster, which then chooses where to deploy the user's workload.

Description L2 – Cluster

Level 2 deals with the configuration and management of a single cluster. This includes components such as the container orchestration system, node configurations and system resources. In the context of this project, the orchestration system chosen to manage and coordinate containerized applications is K8s. Leveraging K8s brings advantages that significantly contribute to the efficiency and robustness of our architecture.

Description L3 – Server/Node

Level 3 focuses on the management of resources within a single node, to ensure that applications run efficiently and reliably, and that information can be collected and used to optimise resource usage. A node is divided into:

- *HW*: Hardware refers to the physical components of the computer system, such as CPU, RAM, storage devices and network interface.
- *OS*: The operating system is a software layer that sits on top of the hardware, manages resources such as memory, CPU, and I/O devices, and provides a range of services such as file systems, networking, and security.
- *Container management*: A container runtime is a software layer that allows applications to run in isolated environments called containers. Containers provide a lightweight and efficient way to package and deploy applications, along with their dependencies and configuration. The container runtime manages the lifecycle of containers, including creation, launch, shutdown, and deletion. Policies for container management can be configured at this layer. In GLACIATION, the container that is executed depends on the operations required for the specific use case.
- *Knowledge graph*: is used to collect and organise information about resource usage and application status. This allows for a comprehensive and global vision of the cluster and can aid in making informed decisions on how to optimize resource usage. It is worth noting that each platform has its own dedicated Platform Knowledge Graph, providing an aggregated snapshot of system health for that specific platform maintaining a distributed fashion.
- *Platform Functions*: the management system provides several features such as swarm agents, ML techniques and APIs to help manage different aspects of the platform.

2.3.2. Reference Architecture

Figure 7 shows the Logical View of the overall architecture. As mentioned earlier, as multi-site execution is not required, although there is the design for 'execution offloading' to edge devices, in the implementation, integration and testing phases everything is done on a single cluster basis. Finally, Figure 8 shows the detail of the component called GLACIATION Node.

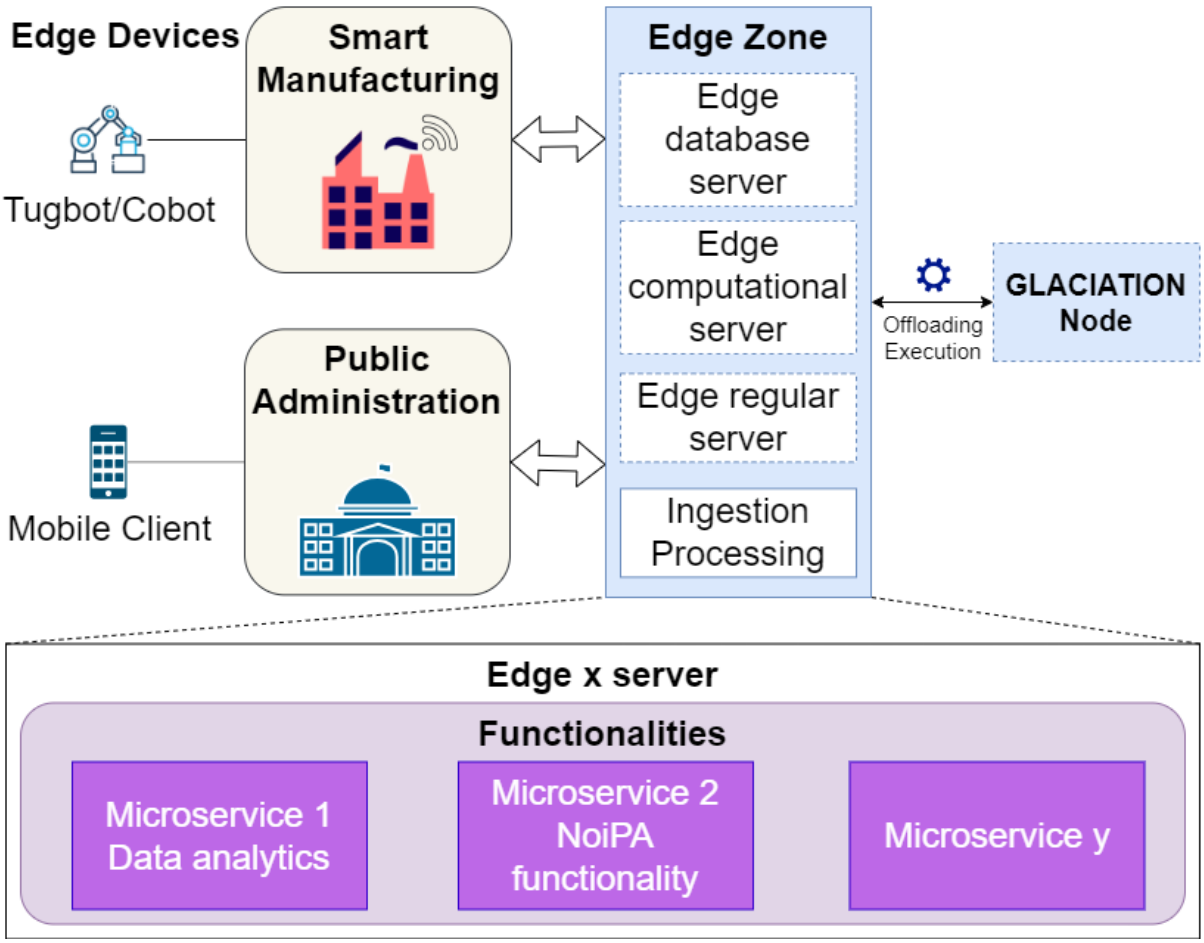

Figure 7: Architecture Logical View.

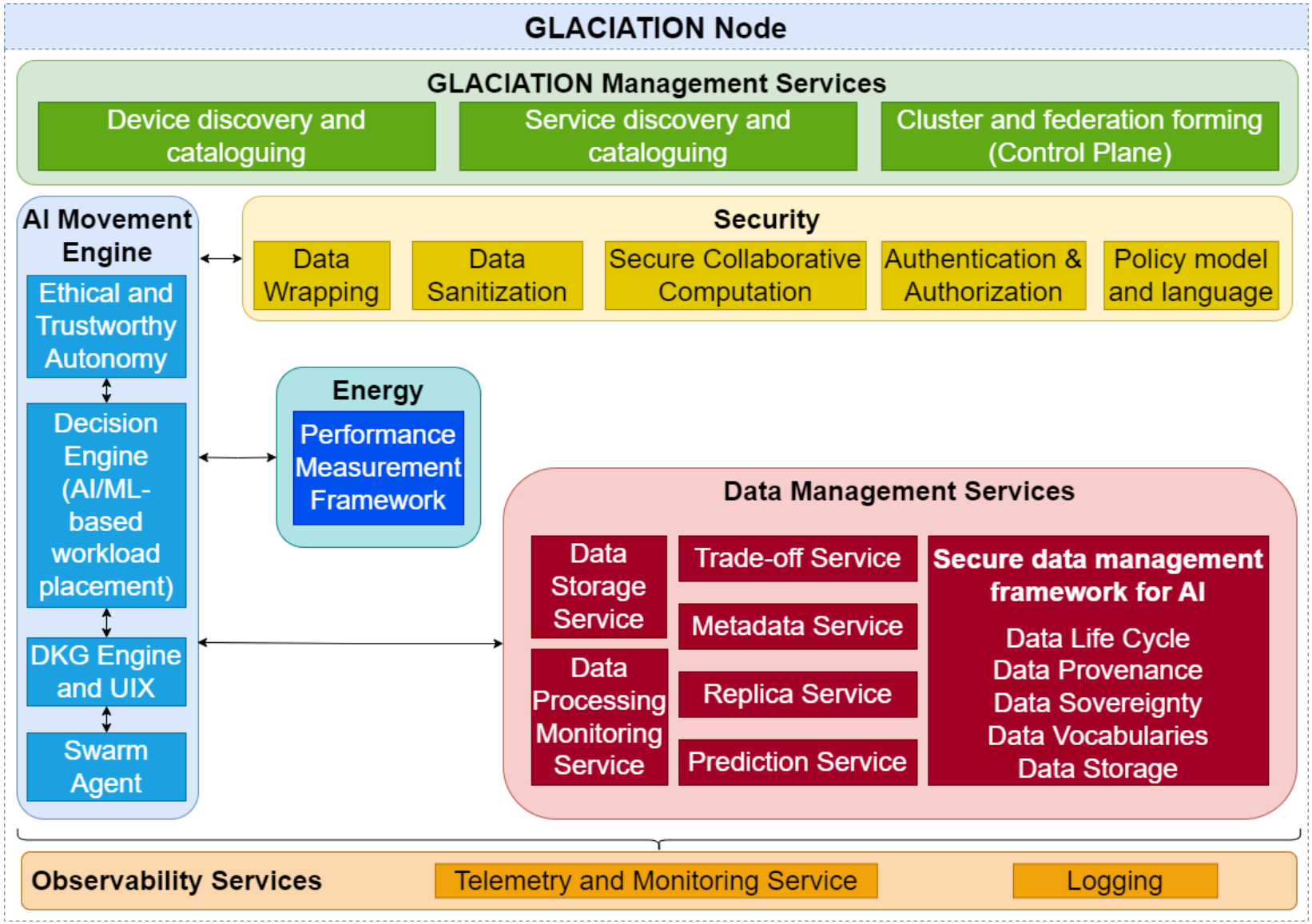

Figure 8: Architecture Logical View: GLACIATION Node detail.

The Reference Architecture above displays the several elements that were determined after going through many stages of use case interviews, discussions, and requirement elicitation. The macroblocks AI Movement Engine, Security, Energy, Data management, Observability, and Management services are where they are categorized. Here is a list and quick description of it:

- Edge devices collect a wide range of data for various situations and use cases, including public administration, manufacturing, and corporate sectors. For instance, in the manufacturing sector, data could be related to the movements and functions of robots and cobots. The data management flow starts here when the data travels through edge servers. An element of the **ingestion process** that distinguishes between data and metadata is used to identify and normalize the data.

- The GLACIATION system performs data movement and workload placement efficiently and securely, thanks to the **decision engine**. Therefore, this component is the primary one receiving task requests to the system, making smart decisions based on the cluster status. It will operate and update the information contained in the DKG, also known as the Platform Knowledge Graph, when dealing with a single slice. The deployment scheduler of this decision engine incorporates machine learning and artificial intelligence algorithms, meticulously designed to adhere to principles of trustworthiness and ethics.

- The management of the DKG involves the utilization of supportive services, diverse queries, and a UI/UX for visualization, which aggregation results in the creation of what we can call **DKG Engine**. These queries enable effective interaction with the graph, allowing users to retrieve, analyse, and manipulate the data stored within. The UI/UX component provides an intuitive and user-friendly interface for users to interact with the DKG, facilitating seamless navigation and exploration of the graph's content.

- A **swarm agent** efficiently manages the transfer or the distributed searching of the DKG state between the various nodes in the cluster. To facilitate this process, it utilizes advanced algorithms, including the Ant Colony Optimization algorithm (ACO). ACO is a probabilistic technique that is widely used for solving computational problems that involve finding optimal paths through graphs. In the context of swarm intelligence, artificial ants represent individual agents inspired by the behaviour of real ants. The communication mechanism in ACO is based on the concept of pheromones, which are chemical substances that ants deposit on their trails to communicate with other members of the colony. Similarly, in the swarm-based component, pheromone-based communication serves as the predominant paradigm for coordinating the distributed querying and transferring of DKG states among nodes in the cluster.

- An **energy framework** component analyses energy metrics and provides data for making decision.

- Complete control over all data-related aspects is exerted at the **data management** level. This includes tracking data provenance, using standardized vocabularies, prediction services, storage techniques, replication procedures, and data sovereignty issues.

- To ensure robust security measures, the GLACIATION system incorporates various components specifically designed to address data protection and access control, visualizable in the diagram in the **security** macroblock. These components play a crucial role in safeguarding sensitive information and preventing unauthorized access. Such functionalities are implemented in the **data sanitization** and **data encryption** modules, which employs advanced techniques, including differential privacy. By applying these techniques, the system ensures that data is transformed in such a way that individual records remain anonymous and protected against de-anonymization attacks. In addition to data protection, the system also implements robust **access control and authentication** mechanisms, responsible for verifying the identity of users and ensuring that only authorized individuals or entities can access specific resources or perform certain actions within the system. This architecture ensures that potentially malicious code executed within the system is contained and unable to compromise the integrity or security of the underlying infrastructure. By restricting access to system resources, the system minimizes the risk potential vulnerabilities that could be exploited by malicious actors.

- Within the GLACIATION system, a set of platform **management services** is in place to ensure efficient orchestration and control of the overall platform operations. These services play a crucial role in enabling integration, microservice communication, and scalability. One of these services is device discovery and cataloguing. A service responsible for identifying and registering devices in the cluster gaining visibility into the available resources and capabilities offered by each device. This information facilitates then effective resource allocation and workload placement. Another important management service is service discovery and cataloguing. This service allows the system to automatically discover and register various services and functionalities provided by different components. This enables the microservice intra-communication maintaining their corresponding locations in the distributed system. It represents the base networking mechanisms for the dynamic service composition. In the end, cluster federation plays the role of integration and coordination of multiple independent servers to form a unified and cohesive cluster. This enables the management and control of distributed resources as a single entity. Federation allows centralized governance, resource pooling, load balancing and fault tolerance, ensuring high availability of the overall system.

- The GLACIATION system also incorporates **observability services** to ensure comprehensive monitoring and analysis of its microservices architecture. These services play a vital role in extracting telemetry and monitoring containerized components, enabling real-time insights into the system's performance and health. By collecting and analysing telemetry data, the observability services facilitate proactive system management, identifying potential issues and optimizing resource utilization.

2.4. Deployment View

The Deployment View provides an overview of how the system's components are deployed and distributed across different hardware or software environments. It focuses on the physical deployment and allocation of software components, including their execution platforms and their interactions with the underlying infrastructure.

The below diagram illustrates the Deployment View for GLACIATION, showcasing the management of components within each cluster using the K8s orchestrator. The system is designed to leverage K8s for efficient deployment and management of its components, ensuring scalability, resilience, and ease of maintenance.

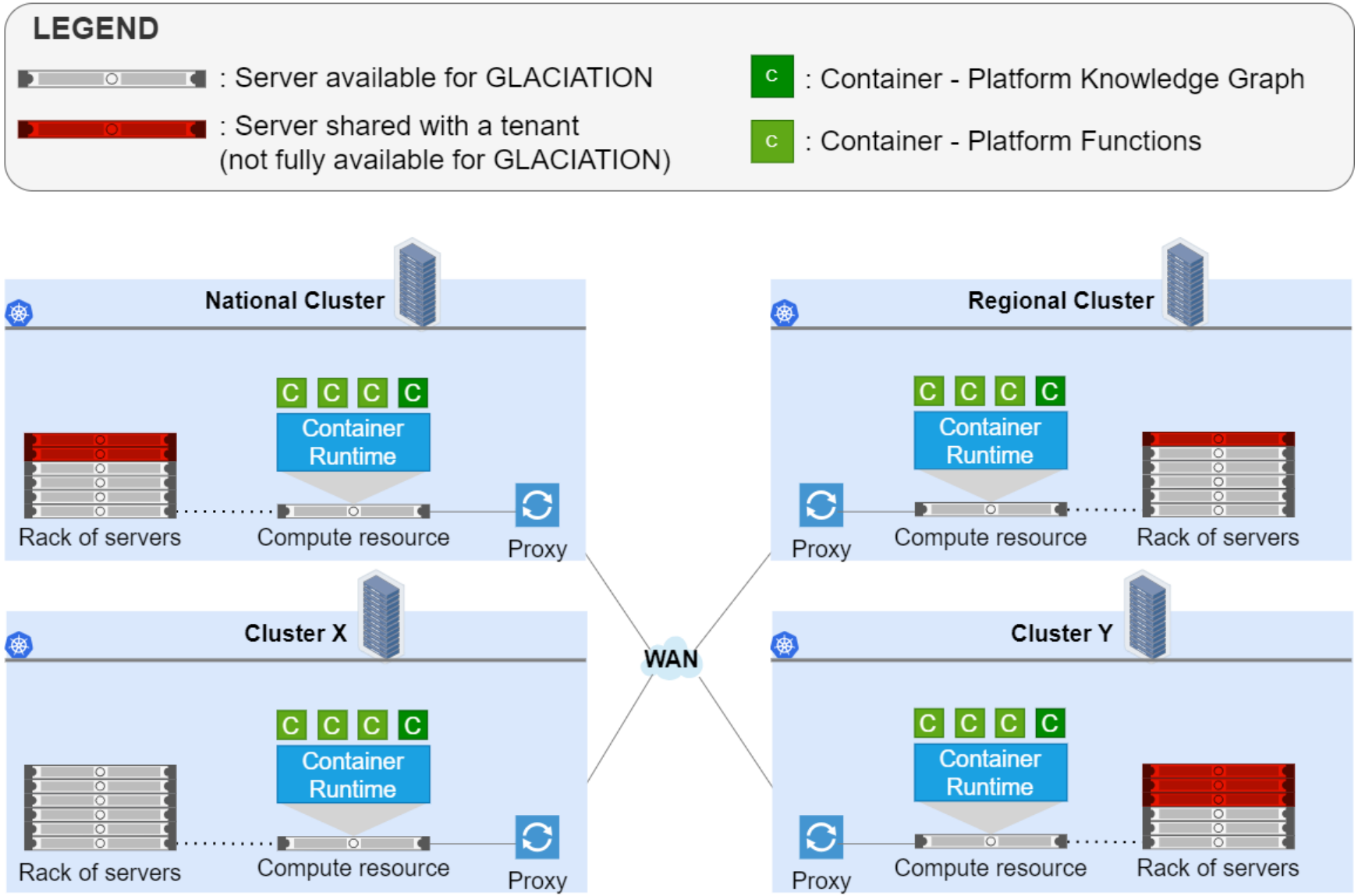

Figure 9: Architecture Deployment View.

The diagram depicts multiple clusters, each representing a logical grouping of nodes or servers. Within each cluster, several components are deployed and managed by K8s: the platform functions and the platform knowledge graph. In scenarios where multi-tenancy is a pivotal aspect, GLACIATION implements robust security measures to ensure the isolation and integrity of tenant-specific data and services. The system's architecture is designed to support multi-tenancy, allowing for the concurrent hosting of multiple tenants on a shared infrastructure. The multi-tenancy is represented by the red and white servers in the diagram.

As mentioned above, although designed, there is no requirement for multi-site at present: consequently, the test lab is set up with one cluster.

2.5. Data Flow Diagram

Having reached this point, the system under development is a distributed microservice architecture that allows analysis jobs to be executed. These jobs are designed to be stored and scheduled immediately or for future use.

Administrators can trigger analysis jobs from various locations, including data centres and factory terminals.

In this last, much more specific view, it was decided to use a DFD, which is a way of representing a flow of data through a process or a system (usually an information system). The DFD also provides information about the outputs and inputs of each entity and the process itself. With this diagram, shown in Figure 10, we can have a more specific view of how the components interact with each other. Figure 11 shows details related to “AI Decision Engine” and “DKG Engine and UIX” components.

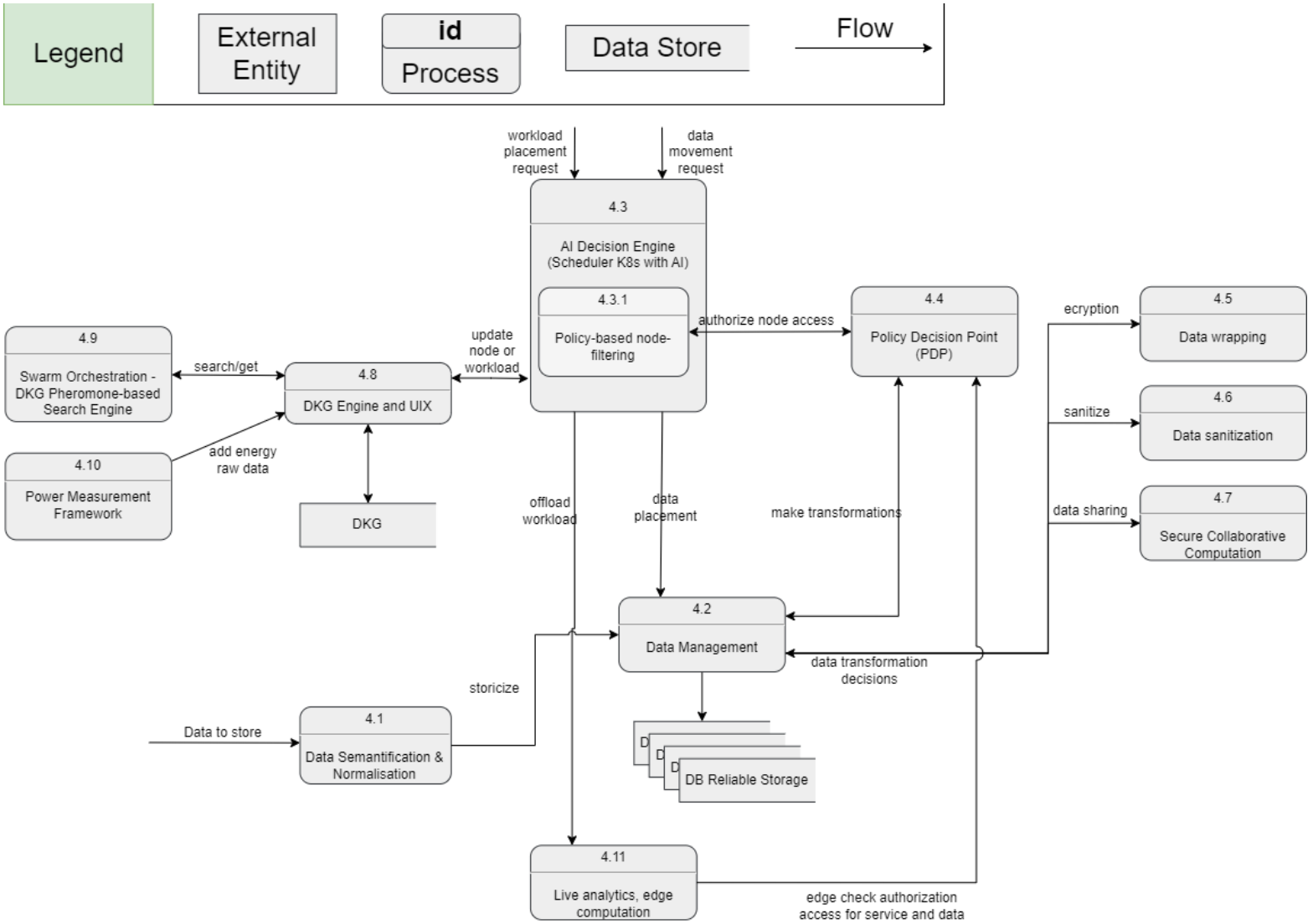

Figure 10: Architecture Data Flow Diagram.

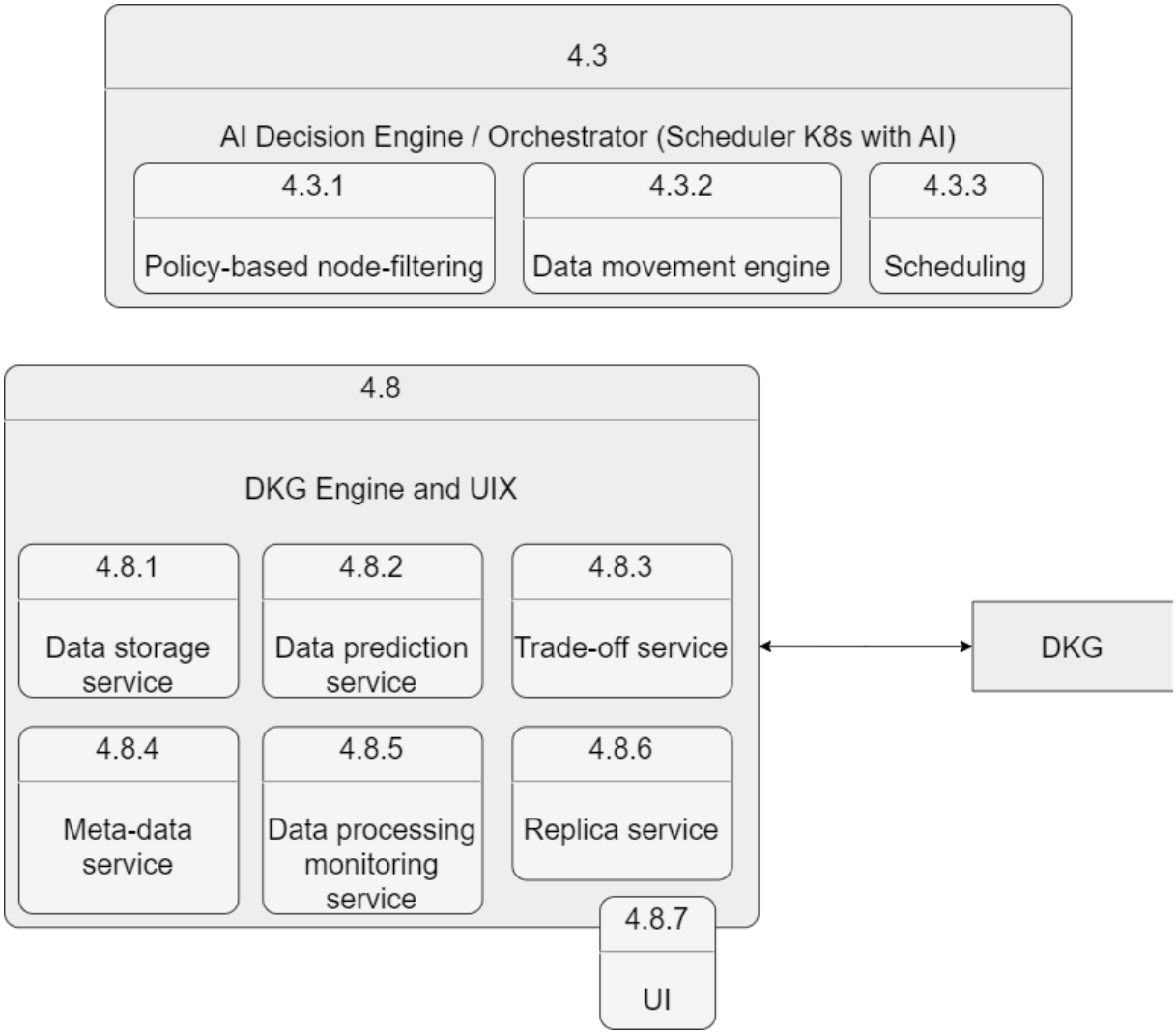

Figure 11: Architecture Data Flow Diagram details.

A first essential functionality is offered by the AI Decision Engine, which receives requests for load placement or data movement and performs various activities. Taking into account a workload placement request, this component extrapolates information such as the required CPU and/or RAM, analyse the information on the current state cluster in terms of both performance and available energy, and then through artificial intelligence it is able to generate a list of potential nodes on which to deploy the workload. But that's not all, communication with the Policy Decision Point is necessary because security policies require filtering of nodes, as not all of them are usable for deployment due to rules inherent to the workload to be deployed. The AI Decision Engine is supported by an advisor system for the final decision. This advisory component uses swarm orchestration techniques and is called SLA Advisory Board. Once the node has been identified, information's on the new state of the system and the running workload are added to the DKG, which will then have an update of the total state of the system. The workload is finally managed through K8s, maintaining a high level of security, energy consumption and control. For data management, the Data management module acts as an interface to the vast heterogeneity of databases that may exist in the final system, providing a single point of contact so as not to be bound to the interfaces of each db. As already mentioned, the Policy Decision Point implements mechanisms for authenticating and authorising access

to nodes, applying security policies in cases where there is a need, for example, to transform incoming data. Data wrapping, sanitisation and secure collaborative computation modules precisely perform the roles of applying the necessary transformations on the data, from cleaning by anonymisation to the encryption of the data themselves and the consequent possible collaborative transfer of information.

K8s helps manage the extraction of energy metrics of individual nodes and the extraction of performance metrics such as network, cpu and ram utilisation. The DKG Engine and UIX component includes all microservices to support the management of the DKG. Indeed, it consists of services for data storage, data prediction, trade-off, metadata, data processing, replication and UI, described in the appropriate section of this document. An important functionality supporting such a distributed database is that offered by swarm techniques and in particular by the DKG pheromone-based search engine, which efficiently searches for and saves information on one of the various DKG slices.

The last functionality offered is Data Semantification & Normalisation, which is used to normalise data received from outside into a format inherent to the GLACIATION system.

The aggregation and seamless communication between these components allow GLACIATION on reaching its objectives in an effective manner.

3. AI Movement Engine

AI movement engine consists of various components fulfilling the requirements of the project. The workload placement engine is responsible for placing incoming workloads on proper worker nodes of a cluster to satisfy a global objective such as minimizing the energy consumption. The Swarm Agent is responsible for retrieving data from the distributed knowledge graph DKG using a swarm-based distributed querying mechanism. The DKG itself is a distributed database capable of modelling and maintaining metadata using semantic web technologies that stores the cluster status. It will be accessible though APIs and user interface. When users and applications need to access protected data that is under security, privacy, and other access constraints/polices the data movement engine will decide to either grant or deny access to the data and when granted, to grant the access of the location of request, location of data, or a third location. These locations may be on the same cluster or not; also, may require data movement. All of the components will fulfil and adhere to the requirements of the ethical and trustworthiness framework that will be stablished in the course of the project.

3.1. Workload Placement Engine

Technical description

Work placement engine is a collection of ML-based components that together try to schedule and place incoming workloads onto the available worker nodes in a K8s cluster.

Technology (Internal Architecture)

The unit of scheduling workload is a K8s Pod that is submitted to the K8s admission APIs. The Pod manifest contains information about the resources demanded by the Pod as well as its execution and placement preferences. Workload Placement Engine has a trained model that is plugged into a serving component which is integrated with K8s scheduling pipeline. LUH trains the model for different cluster sizes and workload types based on the UC scenarios. The serving component interfaces with the DKG and potentially the APIs of a cluster-wide telemetry system. Additionally, the serving component needs to receive higher level (multi-cluster) data such as aggregated energy consumption per class of energy source. The serving component receives the Pod manifest and tries to find the best feasible worker node to run it according to the optimization objective. At the moment the main optimization objective is power consumption reduction. The ML model is still under investigation, however, due to the nature of the clusters inter/intra-connection as well as the structure of the DKG, LUH is investigation Graph Neural Network (GNN) models.

Interface and Interactions

Interface with the DKG to obtain data about the clusters, worker nodes, workloads, and their available/allocated resources.

Interface with the telemetry system through APIs to obtain near-real time status of the cluster including resource usage of the workloads.

Interface with K8s to obtain access to the scheduling farmwork and redirect the scheduling requests to the LUH serving component.

Interface with Swarm agents to obtain aggregated data about the overall behaviour/status of the member clusters of a multi-cluster federation.

3.2. Swarm Agent

Technical description

The Swarm Agent is responsible for retrieving data from the DKG. When a request for the DKG is received at a node in the GLACIATION cluster, the Swarm Agent performs a distributed query on the DKG. This component is represented in Figure 10 as component 4.9.

Technology (Internal Architecture)

The component is based on a Swarm Intelligent pheromone-based algorithm called Ant Colony Optimization. The concept is that whenever a query enters a node, a Swarm Agent (Forward Ant) is created. This agent then carries the SPARQL query from node to node, querying the local knowledge graph on each node. To ensure that all results of the query are found, Forward Ants can create clones to explore multiple nodes simultaneously. Whenever a result is found, a Backward Ant is then created, carrying the result back to the original querying node using the same path taken by the Forward Ant. The Backward Ant leaves pheromones on the nodes along its path, indicating where the results were found, facilitating easier retrieval by future Forward Ants (see Figure 12).

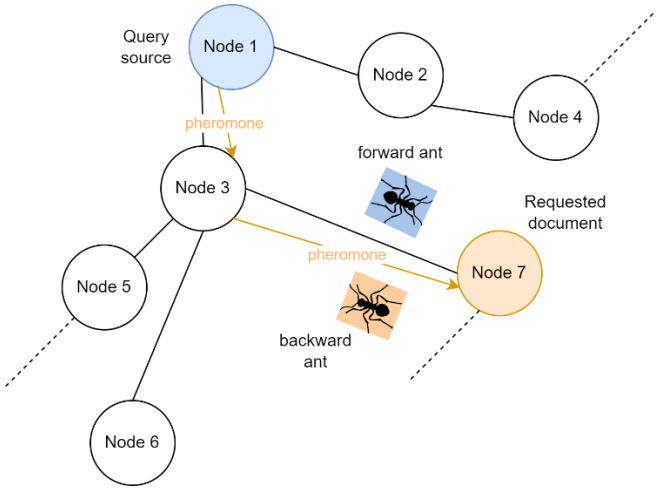

Figure 12: Simplified scheme of nodes, and Swarm Agents going through it following pheromone trails and laying them.

These two types of Swarm Agents, the Forward Ant and the Backward Ant, are currently implemented as Python classes. The Swarm Agent objects can act independently. The nodes of the GLACIATION cluster should be capable of receiving Swarm Agents from their direct neighbours, where neighbours are defined as nodes with a network/wired connection. Each node should maintain two tables: a pheromone table to track pheromone levels and a list storing the history of visited Forward Ants. This history ensures that even if a clone is revisiting, the query is not performed twice.

A more detailed description of how this search engine works is available in deliverable D3.1.

Interface and Interactions

This component interacts solely with the Metadata Service (see Metadata Service) when the service receives a search/get request. When the query results return to the querying node, this component sends an RDF document back to the Metadata Service. In reality, the Metadata Service initiates the first Swarm Agent, sending it out to search for results.

3.3. DKG Engine and UIX

Technical description

UIX component is represented by a low and a high-fidelity prototype of the user interface that defines novel way of representation of a knowledge graph and interactions with knowledge graph. The UIX prototypes are done with figma interface design tool.

Technology (Internal Architecture)

While the UIX prototype is done with figma design tool the real user interface can be done using typical languages and frameworks e.g. typescript/react. The possible way how frontend can be designed and integrated as a component is provided in the figure below:

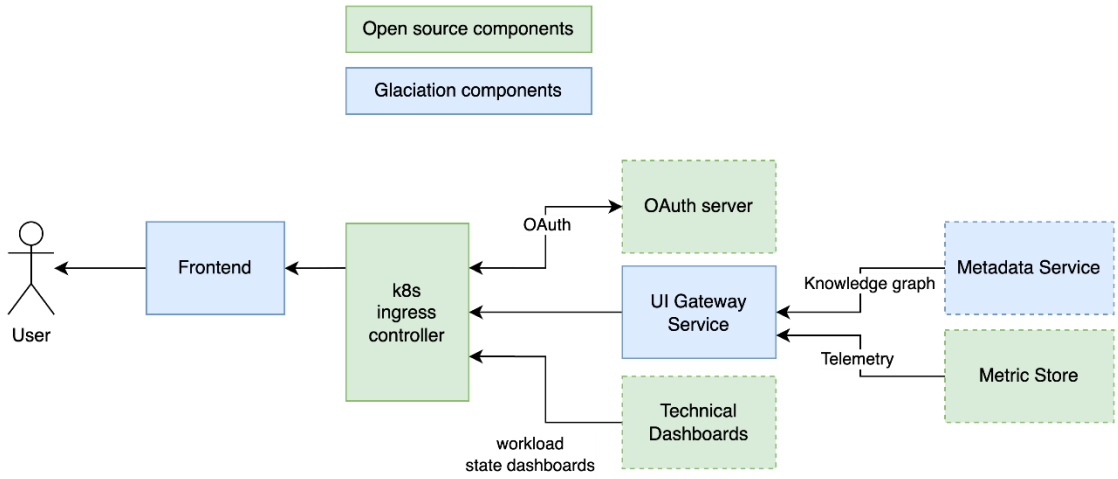

Figure 13: Frontend components design overview and interactions.

There are the following software components:

- GLACIATION Frontend, a client frontend application that implements UIX design.
- K8s ingress controller is the GLACIATION cluster ingress controller. UI traffic entry point, authentication proxy and service router, e.g. envoy gateway[5]. When user is accessing frontend resources, ingress controller redirects user to oauth server and handle auth flow[6].

[5] https://www.envoyproxy.io/
[6] https://kubernetes.github.io/ingress-nginx/examples/auth/oauth-external-auth/

- UI Gateway Service is an intermediary component between the Frontend and other GLACIATION services. The responsibility of the service is to adapt UIX queries into a series of SPARQL requests to Metadata Service and telemetry requests to Metric Store.

Interface and Interactions

Authentication and Authorization – only authenticated and authorized users are allowed to interact with knowledge graph; therefore, every frontend API request is validated against OAuth server.

User interactions with Frontend are converted into REST API calls and forwarded to UI Gateway Service. UI Gateway service abstracts internal components of GLACIATION platform from the frontend. The service transforms high level UI requests into either SPARQL queries to fetch the elements of knowledge graph from Metadata Service or into PromQL language to fetch the metrics from Metric Store.

Additionally, the Frontend exposes technical dashboards with detailed metrics and logs via k8s gateway.

3.4. Ethical and Trustworthy AI

Technical description

This component allows project partners to assess their own software components according to the requirements stipulated by the Ethics Guidelines for Trustworthy Artificial Intelligence[7], which are grouped under the following headings:

1. Human agency and oversight.
2. Technical robustness and safety.
3. Privacy and data governance.
4. Transparency.
5. Diversity, non-discrimination and fairness.
6. Environmental and societal well-being.
7. Accountability

Registered users can create assessments, complete those assessments, and retrieve them later. A dashboard is created for each assessment to illustrate its degree of completion and its score under each requirement.

The component is hosted on an Ubuntu 22.04 LTS server running ASP.NET Core Runtime 8.0[8]. It the still under development.

[7] https://ec.europa.eu/digital-single-market/en/news/ethics-guidelines-trustworthy-ai
[8] https://dotnet.microsoft.com/en-us/apps/aspnet

Technology (Internal Architecture)

The component is implemented as a data-driven web application. The front end is developed using SurveyJS[9], which allows users to interact with forms in a responsive manner. An ASP.NET Core middleware tier contains the application logic and is responsible for data persistence. The back end is a MariaDB[10] relational database.

Interface and Interactions

The user interacts with the web application by means of a web browser. From the home page of the application the registered user can create a new assessment or retrieve and update an existing one. Within each assessment, the user is guided through the various requirements according to which their software component is assessed. Finally, the results of the assessment are presented in the form of a dashboard.

There are no software interfaces with other project components.

3.5. Data Movement Engine

Technical description

The Data Movement Engine (DME) is designed to efficiently manage data placement and movement in a distributed computing environment through the use of swarm intelligence algorithms. This component is essential for optimizing data handling, which in turn, improves system performance and resource allocation.

Technology (Internal Architecture)

The DME leverages Python for the development of its swarm intelligence agents, due to Python's effectiveness in processing complex algorithms and managing data. These agents rely on a Distributed Knowledge Graph (DKG), developed in Work Package 6 (and described briefly in Section 3.3), to make informed decisions about data movement by understanding the relationships and dependencies of data across the system.

Integration with the Secure Data Management Framework for AI (codenamed “Verdelix), outlined in Work Package 3 (and briefly described in Section 6.7), ensures that all data movements comply with stringent security and integrity standards. This framework is vital for safeguarding the data during its transfer and storage, addressing concerns around data protection and compliance.

Interface and Interactions

The Data Movement Engine (DME) is equipped with a sophisticated interface mechanism, designed to facilitate efficient and secure interactions between its components and external systems. At the heart of these interactions lies the implementation of an OpenAPI, which standardizes the communication protocols, ensuring seamless integration and operability across diverse computing environments.

[9] https://surveyjs.io/
[10] https://mariadb.org/

Key Interactions:

1. Data Source to Swarm Intelligence Agent: Initiates with data sources providing real-time data to the Swarm Intelligence Agents. These agents use Python for analysing and making decisions regarding data movement, utilizing insights from the Distributed Knowledge Graph (DKG).
2. Swarm Intelligence Agent to DKG: The Swarm Intelligence Agent consults the DKG to understand data relationships and determine the best strategies for data movement. The DKG is instrumental in the decision-making process.
3. Decision Making and Instruction Dispatch: Following the decision on the optimal data movement strategy, the Swarm Intelligence Agent issues instructions to the Verdelix framework, detailing the destinations for data transfer.
4. Verdelix Framework to Storage Nodes: This operationalizes the data movement instructions.

Figure 14 provides sample interactions for the components of the DME.

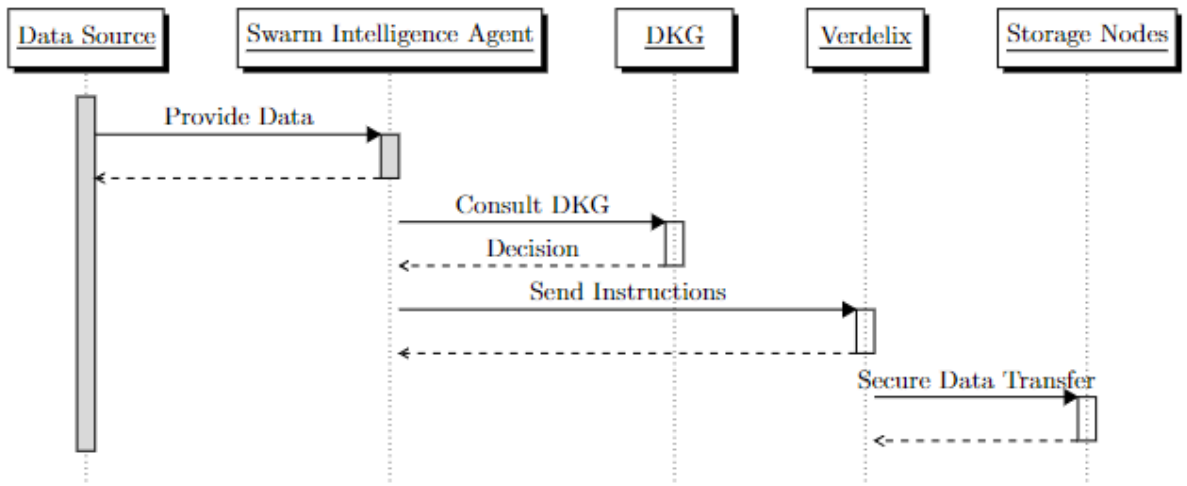

Figure 14: Example of DME interactions.

4. Security

In this section, we introduce a set of components and services to secure the architecture of the GLACIATION platform and provide advanced security features (wrapping and sanitization in particular) to workloads running on it. The section is structured as follows.

Subsection 4.1 illustrates the authentication and authorization mechanisms, which enable the authentication of users and the regulation of their access requests, with consideration of their privileges according to their roles.

Subsection 4.2 illustrates the policy model and language, as well as the identification, integration, and configuration of the admission control service. This service, by intercepting K8s API requests, supports the enforcement of policies, and strengthens governance of the entire K8s cluster.

Subsection 4.3 describes the data wrapping component, which provides transparent server-side encryption of block, file, and object storage. With this component, the difference between an application working with plaintext and encrypted data is as minimal as a one-line change in the configuration of the workload.

Subsection 4.4 describes the data sanitization service, which provides a scalable solution for the anonymization enabling efficient enforcement of k-anonymity and l-diversity requirements for large datasets.

Finally, Subsection 4.5 illustrates the working of secure collaborative computation, providing secure multi-party computation on encrypted data as well as also their possible anonymization through differential privacy.

4.1. Authentication and Authorization

4.1.1. Authentication

Technical description

Authentication refers to the process of proving a user's identity. Usually, users prove their identity by providing their credentials, that is, an agreed piece of information shared between the user and the system. A specific category of credentials, like username and password, are usually called an *authentication factor*. Even if password authentication is the most well-known type of authentication, other authentication factors exist. Typically, authentication factors are classified as follows:

- Something the user knows: this authentication factor requires a user to prove they know something. This is something previously shared between the user and the Identity Access Management (IAM) system.
- Something the user has: the user must prove they have something, such as a mailbox, a smartphone, a smart card. The system presents a challenge to the user to make sure

they have the required authentication factor. A common example is to send a Time-based One-Time-Password (TOTP) to the user's device.

- Something the user is: this authentication factor is based on a piece of information that is inherent to the user. Typically, this information is a biometric characteristic, like fingerprints and facial recognition.

The process of authentication based on just one factor is called single-factor authentication, but it is possible to use additional authentication factors to increase the security of the authentication process.

Technology (Internal Architecture)

K8s clusters have two categories of users: service accounts managed by K8s, and normal users. It is assumed that a cluster-independent service manages normal users, so they cannot be added to a cluster through a K8s API call.

In contrast, service accounts are users managed by the K8s API. They are bound to specific namespaces and created automatically by the API server or manually through API calls. Service accounts are tied to a set of credentials stored as Secrets, which are mounted into pods allowing in-cluster processes to talk to the K8s API.

API requests are tied to either a normal user or a service account or are treated as anonymous requests. This means every process inside or outside the cluster, from a human user typing kubectl on a workstation, to kubelets on nodes, to members of the control plane, must authenticate when making requests to the API server, or be treated as an anonymous user.

Interface and Interactions

We do not plan to identify a single one-size-fits-all solution for the authentication of normal users, but rather support multiple authentication modes according to the availability of the K8s API. K8s supports the following authentication strategies:

- Static token file: bearer tokens are statically configured in a CSV file indicating token, username, user id followed by optional group names. Tokens last indefinitely, and the token list cannot be changed without restarting the API server.
- X509 client certificates: use one or more certificate authorities to validate client certificates. When the client certificate is presented and verified, the common name of the subject is used as the username for the request, the organization fields are used to indicate user's group memberships.
- Open ID Connect: flavour of OAuth2 that returns an additional field with the access token called an ID token. This token is a JSON Web Token (JWT) with well-known fields signed by the identity server. The authenticator can use it to identify the user holding it without interactions with the identity provider, which provides a very scalable solution.
- Authentication proxy: the API server can be configured to identify users from request header values (e.g., X-Remote-User), when these values are set by an authentication proxy holding a valid client certificate (i.e., a client certificate signed by the CA of the API server)

- Webhook: when a client attempts to authenticate with the API server using a bearer token, the authentication webhook POSTs to the remote service an object containing the token. The remote service is expected to evaluate the token and respond with status information to indicate the success or failure of the login.

Depending on the specific requirements of the specific use cases, a different authentication strategy can be used.

4.1.2. Authorization

Technical description

Authorization is the process of giving someone the ability to access a resource. There are several different authorization strategies that computer systems leverage during application deployment. The most prominent ones are Role-Based Access Control (RBAC) and Attribute-Based Access Control (ABAC).

RBAC treats authorizations as permissions associated with roles and not directly with users. A role is nothing but a collection of permissions. Managing authorizations with RBAC is simple because system managers can deal with users and permissions in bulk instead of having to deal with them one by one.

Technology (Internal Architecture)

K8s authorizes API requests using the API server. It evaluates all the request attributes against all policies and allows or denies the request. All parts of an API request must be allowed by some policy to let the request through. This means that permissions are denied by default. A deny returns an HTTP status code 403.

Interface and Interactions

The K8s API server may authorize a request using one of several authorization modes: Node, ABAC, RBAC, and Webhook. Among these options, we have identified RBAC as the most mature and well-integrated with the functioning of K8s. It uses the rbac.authorization.k8s.io API group to drive authorization decisions, allowing admins to dynamically configure permission policies through the K8s API.

4.2. Policy model and language

4.2.1. Admission control service

Technical description

This service validates the creation, modification and deletion of K8s resources within the K8s cluster with the goal of meeting governance and legal requirements, but also ensuring adherence to best practices and institutional conventions.

For example, we could enforce policies like:

- All images must be from trusted sources and with an explicit image tag version.
- All pods must have resource limits.
- Do not allow ingress resources intercepting the traffic of the entire cluster.
- Restrict the set of external IPs to an allowed list of IP addresses.
- Enforce restrictions on the use of privileged pods.

Technology (Internal Architecture)

K8s allows decoupling policy decisions from the inner workings of the API server by means of admission controller webhooks, which are executed whenever a resource is created, updated, or deleted. This service provides mutating and validating webhook that modify objects sent to the API server to enforce custom defaults or reject requests to enforce custom policies.

The admission control service is built-upon Gatekeeper, a customizable cloud native policy controller that is integrated by design with K8s and helps enforce policies executed by Open Policy Agent. This empowers cloud administrators with a mechanism to enforce best practices, and compliance with governance and legal requirements.

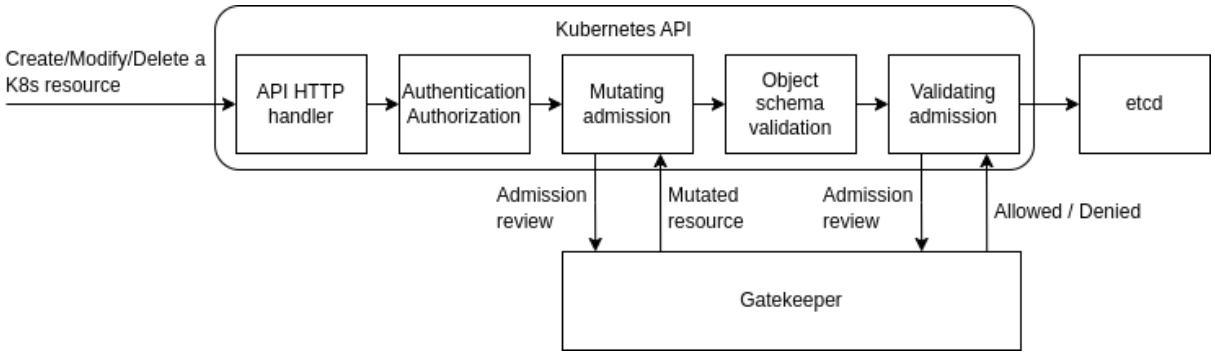

Figure 15: High level architecture of how Gatekeeper integrates with the K8s API.

Interface and Interactions

Being Gatekeeper tailor-made for K8s, it extends the K8s API with new Custom Resource Definitions (CRDs) supporting create, read, update, and delete operations of policy templates and policy enforcements. ConstraintTemplate declares the template of a constraint by stating

the Rego policy that enforces the constraint and the schema of its parameters. Constraint enforces a ConstraintTemplate by specifying the kind of resources affected and the enforcement parameters. For a more detailed view of Gatekeeper and its interface, it is possible to access the official documentation[11].

4.3. Data wrapping

4.3.1. Server-Side Object and Volume Encryption

Technical description

Transparent encryption of object store and K8s volumes offers a robust shield to protect sensitive data at rest. This safeguards information from unauthorized individuals who gain physical access to storage disks or underlying nodes, significantly reducing the risk of data breaches, and ensuring the confidentiality of information at rest. Indeed, in the unfortunate event of a security incident, the encrypted data remains inaccessible, limiting the impact of the breach and protecting sensitive information. This plays a crucial role in aligning with various industry standards and regulations. Indeed, General Data Protection Regulation (GDPR) mandates robust data protection measures for personal data held by organizations within the European Union (EU) and the European Economic Area (EEA).

While encryption adds a layer of security, careful implementation with optimized encryption libraries and hardware acceleration ensures minimal performance impact. This maintains system efficiency, avoids disruptions to daily operations, and keeps energy consumption under control.

By strategically employing server-side encryption within the GLACIATION architecture, organizations gain a multifaceted advantage. Not only it does safeguard sensitive data, but it also simplifies compliance, streamlines security management, and enhances operational efficiency, ultimately fostering a more secure and trustworthy environment.

Technology (Internal Architecture)

To seamlessly integrate server-side encryption within K8s, the target orchestration system of the GLACIATION platform, we rely on the use of open-source cloud native distributed storage systems. Prominent examples are Longhorn, OpenEBS, Piraeus, and Rook. These projects are supported by the Cloud Native Computing Foundation (CNCF) and embody years of efforts and development to provide resource-efficient production ready management of persistent storage in K8s. Based on their maturity levels[12], Rook and Longhorn have an edge over the others, as they are the most stable and used in production environments.

Rook is a complete solution managing file, block, and object storage. It turns distributed storage systems into self-managing, self-scaling, self-healing storage services. Rook orchestrates the Ceph storage solution, with a specialized K8s operator to automate

[11] https://open-policy-agent.github.io/gatekeeper/website/docs/howto/
[12] https://www.cncf.io/project-metrics/

management. It ensures that Ceph will run well on K8s and simplify the deployment and management experience.

Longhorn is a newer project with the goal to deliver simplified, easy to deploy and upgrade, 100% open source, cloud-native persistent block storage without the cost overhead of open core or proprietary alternatives. Moreover, Longhorn's built-in incremental snapshot and backup features keep the volume data safe in or out of the K8s cluster.

Another prominent solution in the storage landscape is MinIO. This solution provides scalable and high-performance object storage (released under dual license GNU Affero General Public License v3.0 and MinIO Commercial License). MinIO supports a complete range of functionality including object locking, retention, legal holds, governance, and compliance. It provides an Amazon Web Services S3-compatible API and supports all core S3 features. MinIO is built to be deployed everywhere – public or private cloud, bare-metal infrastructure, orchestrated environments, and edge infrastructure.

Depending on the requirements put forward by the integration of all the components of the GLACIATION platform, we plan to support either the use of an all-in-one solution like Rook or the two distinct, but easier to use solutions, like Longhorn for block storage and MinIO for object storage.

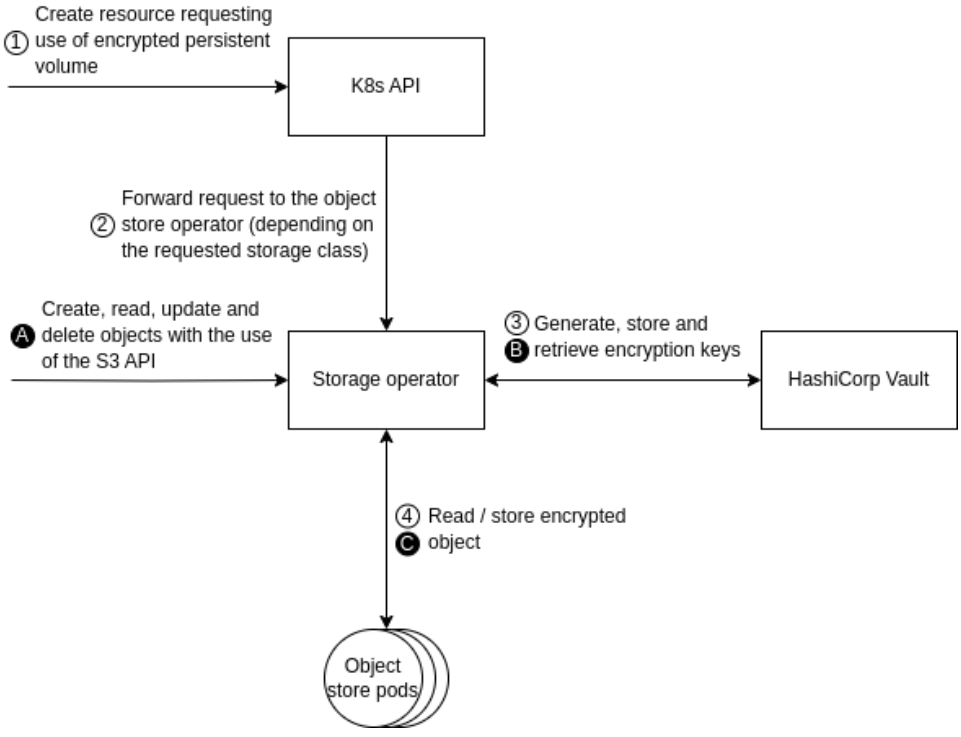

Figure 16: High level architecture to provide server-side object and volume encryption.

Rook, Longhorn, and MinIO support the implementation of transparent encryption of storage resources, but for advanced and more secure setups a Key Management Service is preferred (where supported). HashiCorp Vault is the standard de-facto open-source solution when it comes to secrets management, encryption-as-a-service, and privileged access management. Vault provides comprehensive audit logging and detailed information about who accessed what secrets and when, aiding in security investigations and compliance. Specifically, its Key

Management secrets engine provides a consistent workflow for distribution and lifecycle management of cryptographic keys.

As shown in Figure 15, when the cluster user interacts with the K8s API to request an encrypted persistent volume, the request is forwarded to the storage operator. The storage operator then takes care of dynamically provisioning the storage resource by interacting with Vault for the generation of the encryption key. Following operation on the volumes are encrypted without requiring any change in the application lifecycle. Indeed, it is the responsibility of the storage operator to retrieve persistent encrypted volumes bound to the application and communicate with Vault to retrieve the encryption key.

Other interfaces expose to cloud administrators and cloud applications the features necessary to create new object store buckets and interact with the object store itself.

Interface and Interactions

The interfaces provided by the server-side encryption service target two different audiences: cluster admins and cloud applications.

For clarity, with cluster admins we refer to all those technical roles who have access to the K8s API with permissions to alter the current state of the storage platform. With this regard, block storage providers tightly integrate with the standard K8s resources, so cluster admins only need to declare a Persistent Volume Claim (PVC) belonging to an encrypted storage class. For a complete view of the details regarding persistent volumes, we refer to the official documentation[13]. The same cannot be said for the use of object storage providers, where the specific interface exposed is technology dependent. Rook exposes the creation of new object store buckets as K8s Custom Resource Definitions (CRDs), while MinIO exposes the same functionality with a WebUI and a REST API.

The cloud application interface for the use of block, file, and object storage is unchanged. This is a huge benefit since it translates in no changes to the application code to enhance the security of GLACIATION workloads with server-side encryption.

4.4. Data sanitization

Technical description

Guaranteeing privacy in datasets containing possible identifying and sensitive information requires not only refraining from publishing explicit identities, but also obfuscating data that can leak (disclose or reduce uncertainty of) such identities as well as their association with sensitive information.

The data sanitization service implements an efficient and effective approach to protect user privacy by obfuscating their identities or sensitive information. Three kinds of transformations are supported: k-anonymity, l-diversity, and the use of both.

[13] https://kubernetes.io/docs/concepts/storage/persistent-volumes/

k-anonymity requires generalizing values of the quasi-identifier attributes (i.e., attributes that can expose to linkage with external sources and leak information on respondents' identities) to ensure each quasi-identifier combination of values to appear at least k times.

l-diversity considers each sensitive attribute in grouping tuples for quasi-identifier generalization so to ensure each group of tuples (whose quasi-identifiers will then be generalized to the same values) is associated with at least l different values of the sensitive attribute.

While simple to express, k-anonymity and l-diversity are far from simple to enforce, given the need to balance privacy (in terms of the desired k and l) and utility (in terms of information loss due to generalization). To do so we implement a multi-dimensional algorithm that provides an efficient and effective approach capable of scaling the execution onto a cluster of nodes. With our approach, anonymization is executed in parallel by multiple workers, each operating on a portion of the original dataset to be anonymized. The design of our partitioning approach aims at limiting the need for workers to exchange data, by splitting the dataset to be anonymized into as many partitions as the number of available workers, which can then independently anonymize their portion of the data.

Technology (Internal Architecture)

The sanitization service is a web service built-upon an Apache Spark application. So, to seamlessly integrate it within K8s, the target orchestration system of the GLACIATION platform, we use the Spark on K8s operator. The operator allows Spark applications to be specified in a declarative manner (e.g., in a YAML file) and run without the need to deal with the Spark submission process. It also enables status of Spark applications to be tracked and presented idiomatically like other types of workloads on K8s.

In Figure 16, the process describing the creation of a new data sanitization job is depicted. First, the requesting application sends the configuration of the anonymization task to the data sanitization service, then the request is sent to the K8s API with the creation of a SparkApplication object. From there, the Spark operator kicks in by submitting the Spark application for execution. This leads to the creation of the Spark application driver and executor pods that take care of the execution of the application.

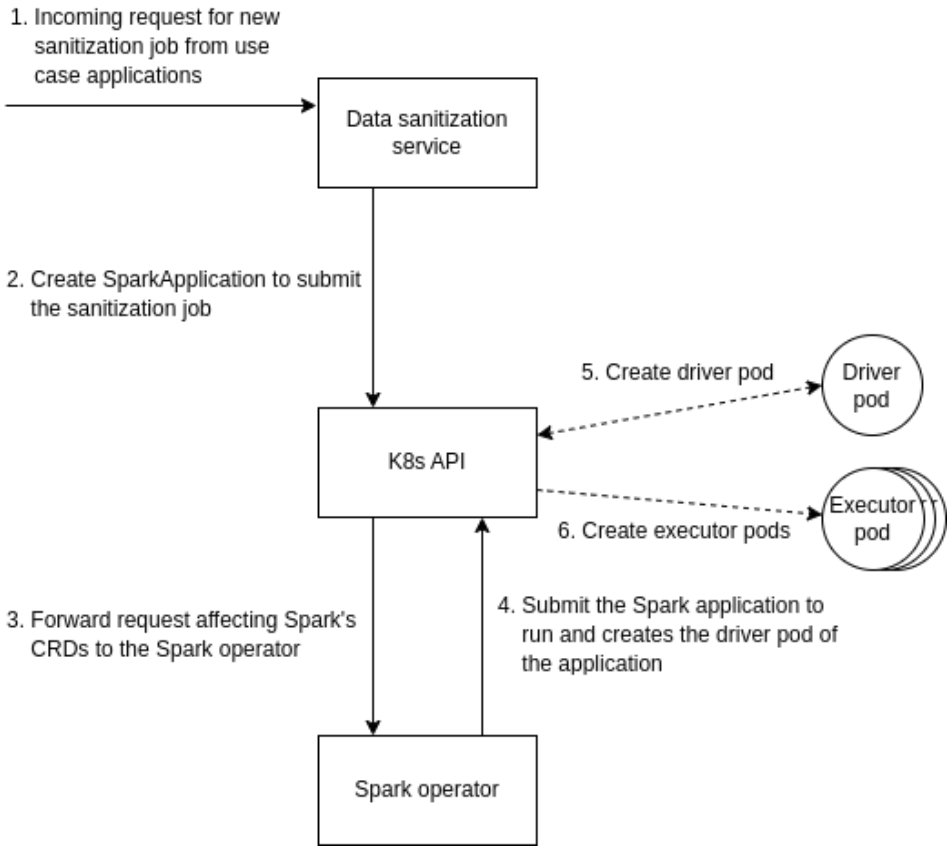

Figure 17: High level architecture showcasing the submission of new data sanitization requests.

The Spark application implementing the anonymization process operates in a distributed manner on the input dataset to produce the sanitized output dataset. We assume these datasets to be stored in object bucket stores. So, we rely on the existence of an object store either within the K8s cluster deployed with a native K8s operator (e.g., Rook, MinIO) or outside the cluster. The interaction with the object store will most likely use the Simple Storage Service (S3) API, the de-facto standard when it comes to interacting with object stores.

Interface and Interactions

Sanitization jobs are configured by the requesting application with numerous parameters with the goal of tailoring the sanitization process according to the specific requirements of the application. These parameters include the input and the output; the classification of attributes in identifiers, quasi-identifiers and sensitive attributes; k-anonymity and l-diversity privacy parameters; the function used for determining the dataset cuts; the set of custom generalization methods for quasi-identifiers (if any); and the list of information loss metrics to compute. Moreover, since the anonymization process distributes the workload among multiple worker nodes, the service also enables the configuration of data distribution and processing. Parameters include the specification of the fraction of the original dataset to be considered in this initial stage, the number of partitions, the fragmentation strategy, and its parallelization and repartitioning scheme. Figure 18 depicts our preliminary REST API.

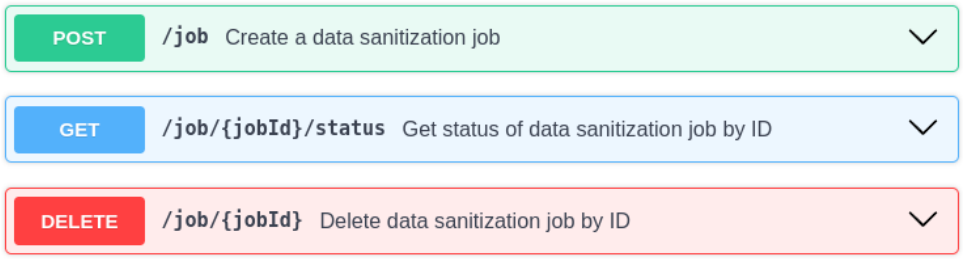

Figure 18: Preliminary REST API of the sanitization service.

4.5. Secure collaborative computation

We distinguish two main components, mainly secure collaborative computation realized with secure multi-party computation (Sec. 4.5.1) and a visual tool to provide initial feedback for privacy parameterization for differential privacy (Sec. 4.5.2). The latter can be useful to foster anonymized data sharing by providing initial visual guidance for the parametrization or in helping to selecting potentially appropriate guarantees for secure computation of differential privacy mechanisms, i.e., computations on encrypted data with anonymized outputs. Next, we detail the associated technologies with their component architecture.

4.5.1. Secure Collaboration

Technical description

To realize secure data collaborations, we protect sensitive business data during the computation with secure multi-party computation (MPC), where multiple computing parties run a distributed computation on encrypted data. While data is already encrypted in transit (e.g., TLS) and at rest (e.g., AES), MPC allows to perform computation on encrypted data without intermediate decryption, i.e., data remains protected even in use.

Thus, MPC allows to evaluate functions on data from multiple parties such that only the output is revealed but the data is never revealed to any party during the process. In other words, MPC replaces a trusted third party that runs a computation on sensitive, unprotected data with a distributed, cryptographic protocol over encrypted data.

Next, we provide an informal description of MPC and refer to D4.1 for more details and references. MPC can be realized via *secret sharing* where a secret input value is split into multiple secret shares that individually hide the input completely, however, a certain number of shares (the so-called threshold) suffice to reconstruct the secret. One can think of secret sharing as a form of encryption and reconstruction as decryption. Typically, MPC runs in a (slower, data-independent) *offline phase* and a (faster, data-dependent) *online phase.* The former phase produces correlated randomness to speed up the latter phase, i.e., it produces random values with certain properties that simplify computation on encrypted data.

MPC typically distinguishes between different types of parties. Specifically, input parties (providing the data for the computation), computation parties (running the computation) and output parties (receiving the computation result). As it fits our collaboration use case, we

assume input parties and output parties are the same, i.e., parties providing the input are also receiving the output.

Technology (Internal Architecture)

To deploy MPC, we leverage as a framework the cloud native MPC stack Carbyne Stack by Bosch Research[14]. It is an open-source and community-maintained project – filling the gap between research frameworks from academia and proprietary solutions from industry. SAP is contributing to the development of Carbyne Stack in the areas of usability and operations[15].

On a foundational level, Carbyne Stack is built from the ground up with cloud-native technology to support scalable, robust MPC solutions. Basically, Carbyne Stack runs as a K8s cluster with services provided in associated pods. It consists of 4 core microservice components called Klyshko, Ephemeral, Amphora and Castor as shown in Figure 19 which is adapted from the Carbyne Stack website[16].

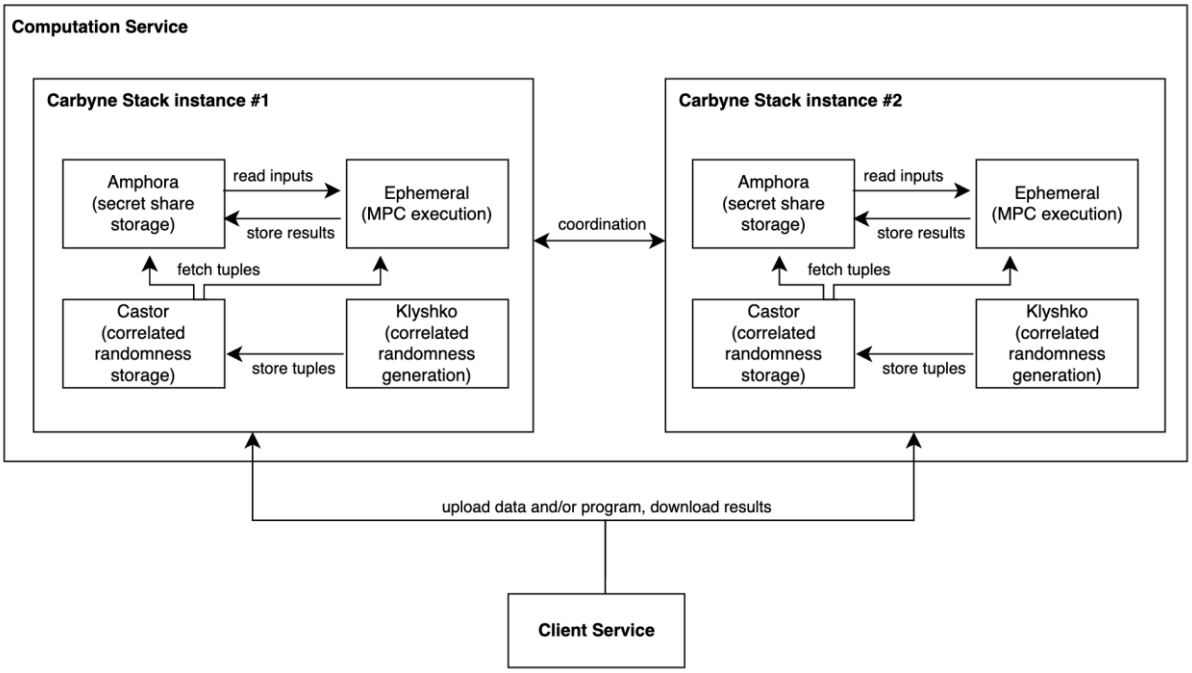

Figure 19: Carbyne Stack components and interactions.

In more detail, the microservice components are as follows:

- **Klyshko** is a service running the offline phase of an MPC computation, i.e., pre-computation of (data-independent) correlated randomness to speed up a later (data-dependent) online phase.
- **Castor** stores the Klyshko-created tuples.
- **Amphora** is the secret store where clients can store the secret shares of their data. It is also used to store and receive the results of the MPC computation.

[14] https://carbynestack.io/
[15] https://carbynestack.io/community/ecosystem/#sap
[16] https://carbynestack.io/getting-started/overview/#carbyne-stack-services

- **Ephemeral** is the service that performs the online phase of the multi-party computation. Ephemeral consumes the tuples from Castor and uses the secret shares from Amphora.

On a higher level of abstraction, building on top of Carbyne Stack, we envision the following services to realize secure collaborations:

- **Client Service**: handles secret-sharing based encryption and decryption for upload and download of inputs and results (from Amphora), respectively.
- **Compute Service**: the computation parties (i.e., Carbyne Stack instances).
- **Coordination Service**: handles registration and coordination of input parties that want to run a collaborative computation and triggers the computation (via Ephemeral) when certain criteria are met (e.g., required number of input parties registered)

These services can run independently of each other. Specifically, the client service should be close to the (sensitive) data to avoid data duplication and can run, e.g., locally at the input party. The coordination service can run at a cloud provider, e.g., on SAP BTP, to facilitate discoverability and coordination of collaboration partners. The compute service is realized by Carbyne Stack.

Notably, the Client and Compute Service suffice to support outsourcing of computation to the cloud while keeping the data protected. Furthermore, it allows collaboration where multiple parties upload encrypted data and trigger a MPC computation on their joint data. However, collaboration requires additional coordination. Specifically, who participates and when should the MPC computation be triggered. While this can be manually solved, e.g., with out-of-band communication or fixed timeslots, we aim to automate this process with an additional service for coordination.

Interface and Interactions

On a lower level, as MPC is a distributed computation, Carbyne Stack provides communication interfaces between running Carbyne Stack clusters, which are the MPC parties, also called “Virtual Cloud Providers” (VCP) in Carbyne Stack. Communication interfaces are provided between VCPs (to run the distributed computation), within the VCP (to interact with the internal microservice components) and from a client to the VCPs (to invoke the services). The client mainly requires the IP addresses of the running VCPs and can use default MPC parameters. For full setup details, we refer to the Carbyne Stack website[17].

On a higher level, the interaction with the VCPs and executing MPC solutions mainly relies on the client. We expose a REST API to allow easy integration and consumption from other components. Our REST API hides the underlying complexities and aims to be as simple and user-friendly as possible.

Our preliminary REST API building on top of Carbyne Stack is as follows.

For the Client Service, see Figure 20.

[17] https://carbynestack.io/getting-started/

Figure 20: Preliminary REST API of the Client Service.

For the Coordination Service, see Figure 21.

Figure 21: Preliminary REST API of the Coordination Service.

Note that the Coordination and Client Service themselves interface with the Compute Service, i.e., the provided services from Carbyne Stack, which we aim to hide from the end-user for a more streamlined and simpler interaction. In other words, user will mainly interact with the Client Service and potentially the Coordination Service. For reference, the Carbyne Stack API specification is available on the Carbyne Stack website[18].

4.5.2. Differential Privacy Visualization

Technical description

To help data analysts with assessing the privacy-utility trade-off inherent to anonymization, we provide a visual analysis tool. The Differential Privacy Visualization (DP-Viz) is run as a service that can be accessed with a web browser. It is a frontend that mainly provides a graphic abstraction with additional evaluations and parameterization evaluations for a backend that performs the actual anonymization. As such, the main interface to DP-Viz is the URL at which the DP-Viz service is running. It provides visual feedback and initial guidance for the anonymization parameterization, helping data scientists to evaluate privacy and utility trade-offs.

In more detail, we focus on differential privacy (DP) – which is a widely deployed privacy notion – and aim to provide guidance for its associated privacy parameters. For detailed description of differential privacy, we refer to D4.1 and give an informal description next. Differential privacy aims to hide the influence any single record (e.g., associated with an individual) can have on

[18] https://carbynestack.io/documentation/reference/api-specification/

a function evaluation. Typically, this is achieved by adding noise, calibrated with privacy parameters and the sensitivity of a function.

The sensitivity of a function is the maximum difference that the inclusion/exclusion of a single record can have on the function output. For example, for a counting query, the inclusion/exclusion of a record alters the result by at most 1, thus, the sensitivity for the count function is 1. Further, there are privacy parameters ϵ (epsilon) and δ (delta) and data sets derived by the inclusion/exclusion of a record are called neighbouring. Epsilon governs how similar (in a distributional sense) noisy outputs are over neighbouring inputs. In other words, how much noise is added, which provides a utility-privacy trade-off. Delta can be interpreted as allowing a negligible probability to reduce the privacy guarantee of differential privacy to improve accuracy. Typically, delta is set to be very small with respect to the size of the data.

Technology (Internal Architecture)

The component relies on the Java version of Google's DP library[19] as a backend to perform the actual anonymization. The frontend is a web interface, i.e., reachable via the Browser, and is built with the following technologies:

- SAPUI5, a frontend toolkit for web-based graphical interfaces with HTML/CSS/Javascript. SAP UI5 uses HTML version 5 and CSS version 3. It is available as a collection of JavaScript libraries for inclusion in a website.
- Plotly.js, an open-source library for visualization, e.g., graphs, charts, and histograms
- Node.js
- SpringBoot

Overall, Java is used for the backend and web technologies (JavaScript, HTML, CSS) for the frontend and the architecture is overviewed in Figure 22.

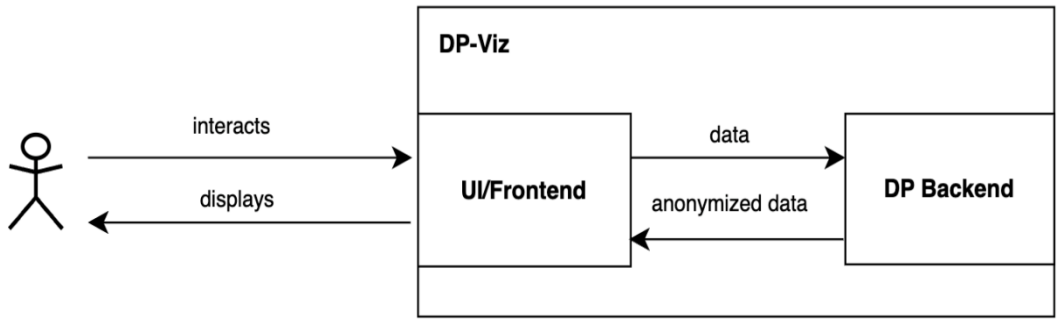

Figure 22: DP-Viz High-level Architecture.

Interface and Interactions

The frontend is reachable via the Browser and provides a graphical interface. The anonymization itself happens in the backend. For interfaces to the DP backend, which we do not expose, we refer to the documentation provided by the GitHub page[20] of the backend. As DP-Viz provides graphical guidance to understand the inherent trade-offs in parameterizing DP, it does not interact with other components.

[19] Available at the following link: https://github.com/google/differential-privacy
[20] Available at the following link: https://github.com/google/differential-privacy

We present the graphical interfaces of the current work-in-progress version of DP-Viz in the following. First, a user can upload a CSV file as shown in Figure 23. Afterwards, the data is displayed per column and a short pre-view of the data is given with possible anonymization parameterizations as shown in Figure 24. Here, a data-independent preview of the noise magnitude is shown. Having finalized the parameterization, the anonymized data is visually compared to the original data as shown in Figure 25.

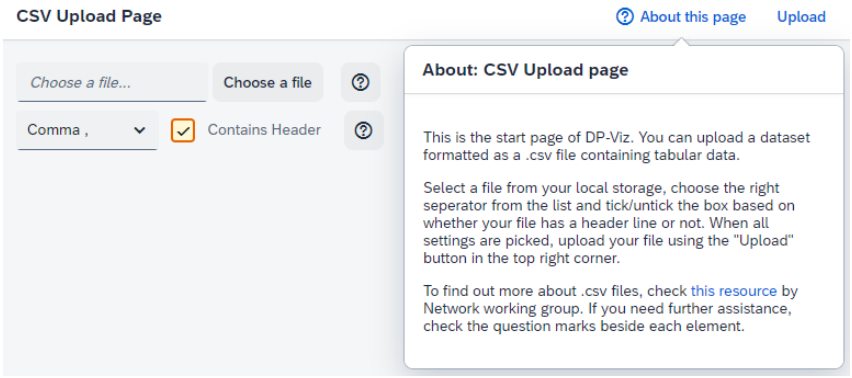

Figure 23: Data Ingestion.

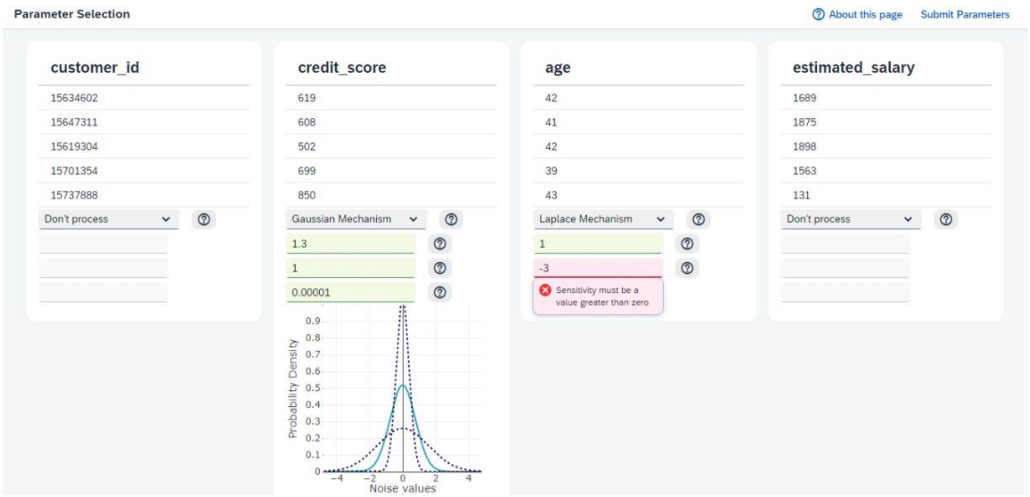

Figure 24: Parameter Selection and Noise Preview with Input Validation.

Figure 25: Analysis Overview - Comparing Utility of Original and Anonymized Data.

5. Energy

This section describes the architecture of two key components that form a part of GLACIATION WP5, namely, the power/performance measurement framework, and the DNA-based data archival system.

5.1. Performance Measurement Framework

Technical description

At the heart of GLACIATION lies the integration of the Power Measurement Framework – a critical component in the pursuit of efficiency and sustainability. By integrating power measurement at multiple levels, GLACIATION takes a granular approach to understanding and optimizing energy consumption. This integration spans the server level, where individual machines are monitored, the rack level, capturing the collective power usage, and the pod level, encompassing the microservices environment.

At the server level, GLACIATION employs power measurement sensors to gather real-time data on individual machine performance. These sensors capture metrics such as CPU utilization, memory consumption, and energy consumption. K8s, acting as the orchestrator, ensures that these measurements are seamlessly incorporated into the overall monitoring and decision-making processes.

Scaling up, GLACIATION extends its power measurement capabilities to the rack level. By aggregating the power metrics from individual servers, the framework gains insights into the collective energy usage within a rack. This holistic view enables efficient resource allocation, optimizing the power footprint of the entire rack.

The microservices architecture, orchestrated by K8s, operates at the pod level. GLACIATION embraces this modularity, measuring power consumption at the granular level of individual pods. This fine-grained approach allows for precise identification of power-intensive microservices, facilitating targeted optimization efforts.

Technology (Internal Architecture)

Our architecture revolves around the development of a robust framework that seamlessly integrates with key technologies used in GLACIATION.

At the server level, embedded with every Dell PowerEdge server, the integrated Dell Remote Access Controller (iDRAC) enables secure and remote server access for out-of-band and agent-free server management tasks. Features include BIOS configuration, OS deployment, firmware updates, health monitoring, and maintenance. One key set of data that iDRAC provides is power usage.

iDRAC is a hardware-based management and monitoring tool developed by Dell for its PowerEdge server line and other Dell EMC server products. iDRAC is designed for secure local and remote server management and helps IT administrators deploy, update, and monitor PowerEdge servers anywhere, anytime. iDRAC provides the rich server power usage data available from Dell PowerEdge servers and the various methods to collect, report, analyse,

and act upon it. It also simplifies server management, reduces downtime, and aids in troubleshooting and maintenance tasks. iDRAC monitors the power consumption in the system continuously and displays the following power values:

- Power consumption warning and critical thresholds.
- Cumulative power, peak power, and peak amperage values.
- Power consumption over the last hour, last day or last week.
- Average, minimum, and maximum power consumption.
- Historical peak values and peak timestamps.

The advanced capabilities of the IDRAC offers an extensive amount of data about power consumption from Dell PowerEdge servers. GLACIATION relies on this for server-level power measurement.

At the rack level, power Distribution Units (PDUs) are devices used to supply power to devices like server, storage and network equipment mounted on a datacentre rack. Smart PDUs are power distribution units which are having remote management capabilities. There are two types of PDUs, the basic type and the smart type. While both can provide reliable power distribution to critical IT equipment within a rack or cabinet, smart PDUs offer several intelligent features to help data centre managers understand their power infrastructure. With data centres becoming more dynamic and complex, smart PDUs have become more prevalent as they offer several key features:

- IP Aggregation: The smart PDUs can be deployed by utilizing units with IP aggregation capabilities. IP aggregation with self-configuration of downstream devices can significantly reduce deployment time and costs.
- Environmental Monitoring: Smart PDUs can incorporate environmental sensors to proactively monitor environmental conditions within the rack to ensure optimal operating conditions.
- Remote Connectivity: Smart PDUs provides the ability to access the PDU remotely through the network interface or serial connection to monitor power consumption and configure user-defined alert notifications to prevent downtime.
- Out-of-Band Communication: If the primary network to the PDU goes down, Smart PDUs provide redundant communications through integration with out of band management devices, such as serial consoles or KVM switches.

GLACIATION uses smart PDUs for measuring rack-level power consumption. The figure below shows one of the APC Metered Rack PDU with 42 output connectors connected to Gateways as a part of GLACIATION Platform.

Figure 26: Smart PDU.

At the pod level, our framework seamlessly integrates with key technologies such as Kepler, K8s, and Prometheus, aiming to provide accurate energy measurements and detailed reporting of power consumption. Kepler is a CNCF project that utilizes software counters and power sources such as RAPL (Running Average Power Limit), ACPI[21] and NVIDIA GPU to measure power consumption by hardware resources within a K8s cluster. By employing an extended Berkeley Packet Filter (eBPF)-based approach, it attributes power consumption to specific processes, containers and K8s pods. One of the standout features of Kepler is its ability to report energy consumption at the pod level. By exposing Prometheus metrics, administrators gain valuable insights into the energy consumption of individual pods.

Prometheus is an open-source systems monitoring and alerting toolkit which collects and stores its metrics as time series data, i.e., metrics information is stored with the timestamp at which it was recorded, alongside optional key-value pairs called labels. Prometheus, with a multidimensional data model, flexible query language, efficient time series databases, supports integration with visualization tools and modern alerting approach, as Figure 27 shows the Prometheus deployment. Prometheus collects rich metrics and provides a flexible querying language; Grafana transforms metrics into meaningful visualizations. Both are compatible with most, data source types and is common for DevOps teams to run Grafana on top of Prometheus.

[21] Available at the following link: http://uefi.org/specifications

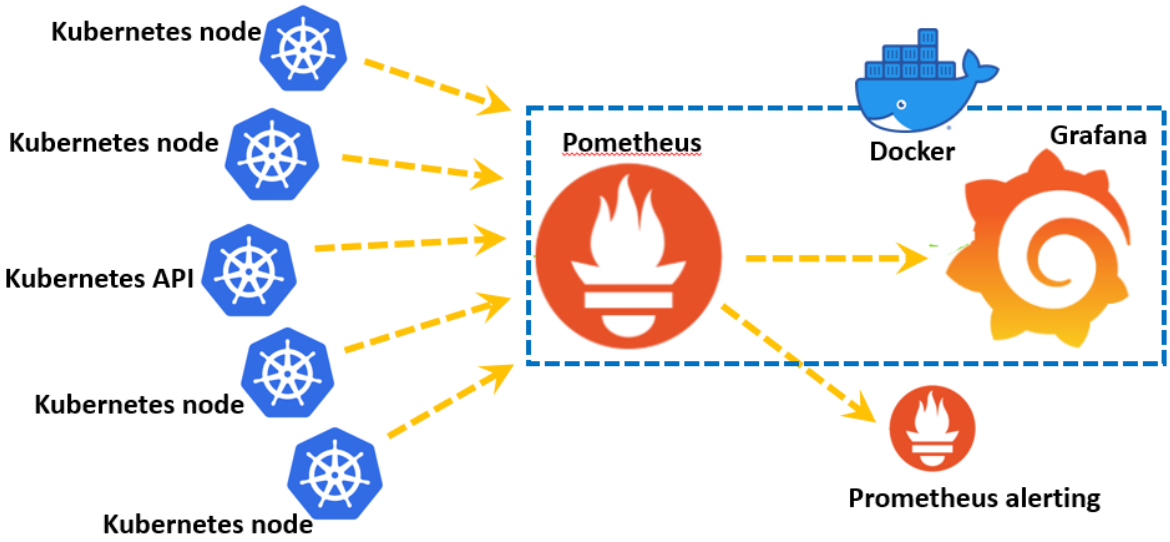

Figure 27: Prometheus-based monitoring architecture.

In order to connect the metrics gathered with iDRAC at the server level and to connect it with other metrics, we integrated the idrac_exporter which is a simple tool designed for use with Prometheus to expose metrics from iDRAC (Dell), iLO (HPE), and XClarity (Lenovo) management controllers. It employs the Redfish API to communicate with these devices and exposes metrics through a /metrics endpoint, which Prometheus can scrape. The software is written in Go and can be downloaded and compiled or used within a Docker container. Configuration options allow specifying the metrics to be exposed, authentication details, and the address and port for binding the exporter. The metrics covered include system power, health, temperature, fan speeds, power supply readings, and system event logs, among others. These metrics are collected on-demand, which may take several minutes depending on the configuration, necessitating a sufficient Prometheus scrape timeout setting.

In order to connect the metrics gathered with PDUs, we developed an exporter that uses Simple Network Monitoring Protocol (SNMP) to pull data from PDUs. Our SNMP Exporter uses Management Information Base (MIB) files to interpret the messages sent by managed PDUs, which is a critical step in network monitoring. MIB is a structure that describes all objects a device can report on, such as CPU, fan, or temperature. MIB is a hierarchical structure, displayed as a navigation tree. Every entry in the MIB tree is a value for a specific component on a specific device. PDUs utilize a separate MIB subtree for querying values via SNMP. These devices can report voltage, amps, volts, kW and kWh. Typically, these values are organized in tables in the MIB. For example, a hierarchical MIB for a rack PDU typically has separate tables for elements such as inlets, circuit breakers and outlets. Our SNMP management system uses these database files to request the agent for specific information and further translates the information as needed for integration into Prometheus.

Interface and Interactions

Putting it all together, a detailed description of the complete flow is visualized in the figure below.

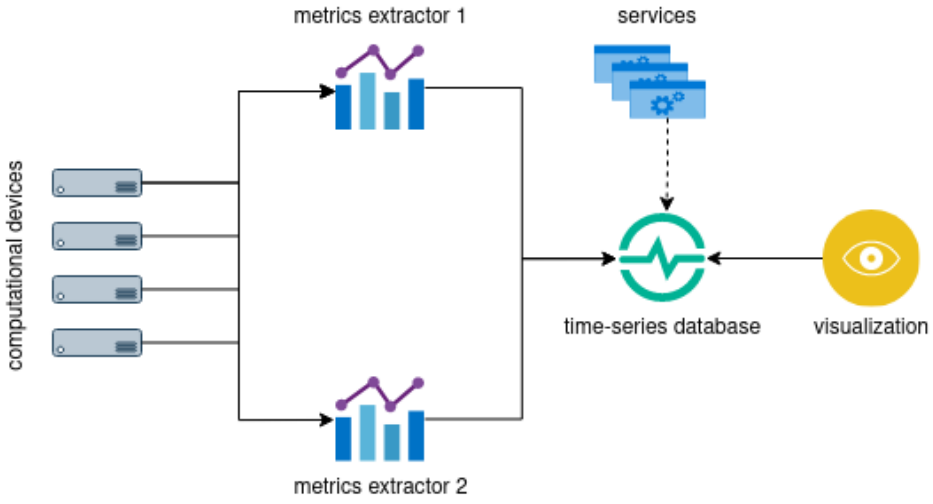

Figure 28: High-level architecture with various components.

The first step in monitoring energy consumption is obtaining relevant data from various sources. Metrics are typically collected by designated metrics aggregators or extractors that are then made periodically available through exporters. The figure shows the process, starting with the finishing machines and moving to the metric extractors. To benefit from different configurations, it is possible to use several metric extractors that can work on different granules with metric exclusion (node level, container level, etc.). and Kepler are used, as explained earlier.

The exporters are then removed when they are normalized, and the measurements are transferred to the time series database (TSDB) for persistence. Prometheus and InfluxDB can be used to store time series data. Finally, effective visualization of aggregated energy measurements is necessary to provide valuable insights. Grafana is a popular platform that provides an easy-to-use interface for creating personalized dashboards and visualizations and works easily with Prometheus and InfluxDB Grafana can create new visualizations in real time by retrieving data from databases into by service monitors. Grafana's service monitors act as an interface between the visualization layer and the time series databases. They specify which metrics should be shown, how data should be distributed to users, and how data should be retrieved from databases. Service monitors can be configured to capture data at regular intervals to ensure visualization is always current with the most up to date metrics.

5.2. DNA-based Long-Term Data Archival

Technical overview

Driven by the promise of data analytics, enterprises have started gathering vast amounts of data from diverse data sources. However, recent studies have pointed out that nearly 80\% of data is ``cold'', or infrequently accessed, and is growing 60\% annually, making it an ideal candidate for archival using cheaper storage devices[22]. Unfortunately, this rapid rate of data growth overshadows density improvement in traditional storage hardware like HDDs and tape.

[22] Intel. Cold Storage in the Cloud: Trends, Challenges, and Solutions. White Pape

Thus, researchers have started exploring the use of hardware--software co-design techniques using existing storage media, or the use of new storage media that has radically different density and durability characteristics compared to HDD and tape.

One such storage media that has received attention recently is synthetic DNA[23]. DNA is a macro-molecule that is composed of four sub molecules called nucleotides (nts) (Adenine (A), Guanine (G), Cytosine (C), Thymine (T)). DNA used for data storage is a single-stranded sequence of these nts. In order to use DNA as an archival medium, digital data is mapped from its binary form into a quaternary sequence of nts using an encoding algorithm. Once encoded, the nt sequence is used to manufacture actual DNA molecules, also referred to as oligonucleotides (oligos), through a chemical process called synthesis that assembles the DNA one nt at a time. Data stored in DNA is read back by sequencing the DNA, which essentially reads out the nt composition of each oligo to produce strings called reads, and then decoding the information back from the reads into the original binary form.

DNA possesses four key properties that make it relevant for archival storage. First, it is an extremely dense three-dimensional storage medium that has the theoretical ability to store 455 Exabytes in 1 gram; in contrast, a 3.5" HDD can store 10TB and weighs 600 grams today. Second, DNA can last several centuries when stored at standard temperature in an anoxic, anhydrous atmosphere; HDD and tape have lifetimes of five and thirty years. Third, it is very easy, quick, and cheap to perform in-vitro replication of DNA; tape and HDD have bandwidth limitations that result in hours or days for copying large EB-sized archives. The fourth property is the difference in Kryder's rate, or the rate at which media density improves every year. Kryder's rate is around 10\% and 30\% for HDD and tape. Thus, if one stores 1PB in 100 tape drives today, within five years, it would be possible to store the same data in just 25 drives. As storage space is a premium commodity in datacentres, using tape for archival storage implies constant data migration with each new generation of tape. A recent article summarized the financial impact of such media obsolescence on the movie industry. DNA does not have this problem as the Kryder's rate for DNA is theoretically zero given that the density of DNA is biologically fixed.

Technology (Internal Architecture)

Given its benefits, synthetic DNA is clearly a promising candidate as an archival medium. However, there are several challenges in designing encoders for DNA storage. First, there are several biological constraints that must be respected during encoding to ensure that DNA molecules can be synthesized and sequenced: (i) Experiments have demonstrated that DNA sequences that have a high number of repeated nts, also known as homopolymers (e.g. TTTT) create problems for sequencing. Thus, the encoder must avoid long homopolymer repeats; (ii) Oligos with a low GC-ratio (fraction of Cs and Gs in the oligo) are known to be unstable, while those with a high GC-ratio are known to have higher melting temperatures and create problems for synthesis and sequencing. Thus, the encoder must maintain GC-ratios in a well-defined range.

Second, current synthesis processes cannot synthesize oligos longer than a few hundred nts. Thus, as a single oligo cannot store more than a few hundred bits at best, it is necessary to fragment the data and encode it across several oligos. As DNA molecule itself has no

[23] Appuswamy et al., OligoArchive: Using DNA in the DBMS storage hierarchy, CIDR 2019

addressing, it is necessary to add addressing information explicitly in the oligo during encoding in order to be able to reorder the oligos later during decoding. Thus, SOTA approaches reserve a few bits per oligo to encode this indexing information.

Third, as mentioned earlier, synthesis and sequencing are error prone. There can be insertion errors, where extra nts are added to the original oligo resulting in a sequenced read being longer than the oligo, deletion errors where nts are deleted resulting in shorter reads, and substitution errors. DNA storage also suffers from a coverage bias. The average number of reads per oligo generated after sequencing is called coverage. Coverage bias refers to the act that some oligos can be covered by multiple reads, and others can be completely missing as they are not covered by any reads. Coverage bias happens due to the fact DNA synthesis itself creates multiple copies of each DNA molecule. On top of this “physical” redundancy, DNA is also amplified using library preparation steps (like Polymerase Chain Reaction) before sequencing. This amplification creates multiple copies of each synthesized DNA molecule, adding further redundancy. As these amplification procedures are stochastic, different DNA molecules get copied at different rates, leading to coverage bias.

In GLACIATION, we have developed several pipelines that overcome all these aforementioned challenges to enable reliable data storage on DNA synthesized with different chemistries (traditional phosphoramidite method or newer enzymatic method). The figure below shows a high-level overview of one of our pipelines OA-DSM which we developed for traditional phosphoramidite synthesis.

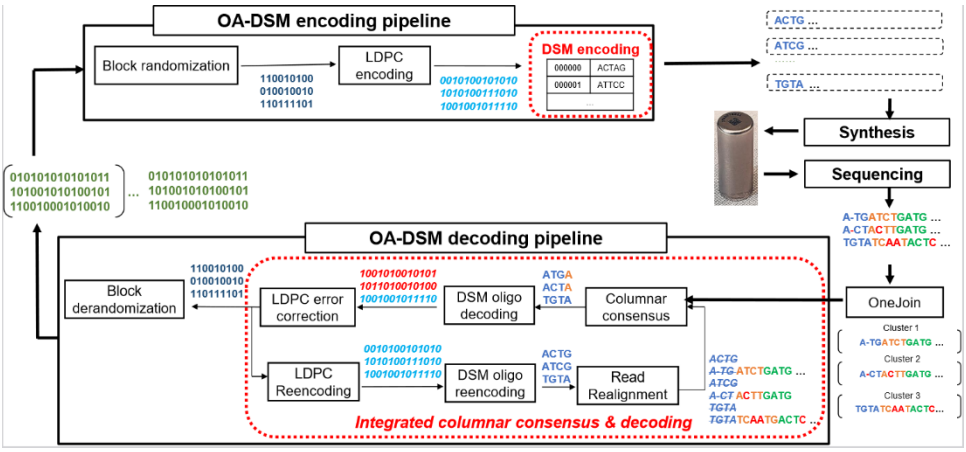

Figure 29: OA-DSM DNA storage pipeline.

The input to the write pipeline is a stream of bits. Thus, any binary file can be stored using this pipeline. The first few steps in OA-DSM are similar to SOTA pipelines. The input data is grouped into blocks of size 256,000 bits. Each block of input is then randomized to improve the accuracy of read clustering in the data decoding stage. After randomization, error correction encoding is applied to protect the data against errors. We use Low-Density Parity Check (LDPC) codes with a block size of 256,000 bits. The LDPC encoded bit sequence is fed as input to the DSM-oligo-encoder which converts bits into oligos. OA-DSM differs from SOTA approaches in this encoding process. SOTA approaches design each oligo as a random collection of nts and map each block of data to a group of oligos, one oligo at a time. As a result of this encoding, a group of oligos becomes the unit of recovery; before data can be decoded, the entire group of oligos must be reassembled by consensus. This is

the reason why SOTA pipelines strictly separate consensus calling from decoding and perform consensus calling first.

The key idea in OA-DSM is to integrate decoding and consensus into a single step, where the error-correction provided by decoding is used to improve consensus accuracy, and the improved accuracy in turn reduces the burden on error correction, thereby providing a synergistic effect. In order to do this, the DSM-oligo-encoder designs oligos using composable building blocks called motifs. Each motif is itself a short DNA sequence that obeys all the biological constraints enforced by synthesis and sequencing. The second major difference of our approach to SOTA is the layout of motifs across oligos which is reminiscent of columnar storage model adopted by modern analytical database engines. OA-DSM-encoder uses the motifs generated from an error-control coded data block to build oligos by adding a new column at a time. This process of column-at-a-time encoding is repeated until the oligos reach a configurable number of columns after which the process is reset to generate the next batch of oligos again from the first column. The generated oligos can then be synthesized to produce DNA molecules that archive data.

As mentioned before, data stored in DNA is read back by sequencing the DNA to produce reads, which are noisy copies of the original oligos that can contain insertion, deletion, or substitution errors. As each oligo can be covered by multiple reads, the first step in decoding is clustering to group related reads together. In prior work, we developed an efficient clustering technique based on edit similarity joins that exploits the fact that due to randomization during encoding, reads corresponding to the same original oligo are “close” to each other despite errors and “far” from the reads related to other oligos. The output of this algorithm is a set of clusters, each corresponding to some unknown original oligo. After the clustering stage, in OA-DSM, we exploit the motif design and columnar layout of oligos to iteratively perform consensus and decoding in an integrated fashion. Unlike other approaches, OA-DSM processes the reads one column at a time. Thus, the first step is columnar consensus which takes as input the set of reads and produces one column of motifs. The choice of consensus algorithm is orthogonal to OA-DSM design. We use an alignment-based bitwise majority algorithm we developed previously for consensus.

The motifs obtained from consensus are then fed to the DSM-oligo-decoder which is the inverse of the encoder, as it maps the motifs into their bit values. Note here that despite consensus, the inferred motifs can still have errors. These wrong motifs will result in wrong 30-bit values. These errors are fixed by the LDPC-decoder, which takes as input the 30-bit values corresponding to one LDPC block and produces as output the error-corrected, randomized input bits. These input bits are then derandomized to produce the original input bits for that block.

As mentioned earlier, SOTA methods do not use the error-corrected input bits during decoding. OA-DSM, in contrast, uses these bits to improve accuracy as shown in the bottom part of the integrated columnar consensus in Figure 29. The error-corrected bits produced by the LDPC-decoder are reencoded again by passing them through the LDPC-encoder and DSM-oligo-encoder. This once again produces a column of motifs as it would have been done during input processing. The correct column of motifs is used to realign reads so that the next round of columnar decoding starts at the correct offset. The intuition behind this realignment is as follows. An insertion or deletion error in the consensus motifs will not only affect that motif, but also all downstream motifs also due to a variation in length.

In our peer-reviewed publication[24], we showed how the approach adopted by OA-DSM mentioned enables reliable data storage on DNA. We have also developed similar pipelines and published them at prestigious venues for enzymatic DNA synthesis[25]. We are continuing to work on several other aspects related to DNA archival, including random access to files stored in DNA, metadata archival, and file format obsolescence issues, to name a few.

Interface and Interactions

We plan to integrate our OA-DSM pipeline with GLACIATION framework to demonstrate the use of DNA as an archival medium in an edge-cloud setting. We plan to develop a REST-based key-value store API that can be called to store and retrieve archival files, with a unique file ID being the key, and the file data being the value. Behind the REST API, we will integrate our OA-DSM pipeline to enable the conversion of input files into oligo strings. However, as DNA synthesis and sequencing are manual biochemical processes, we will not be performing them in reality in the integration. Instead, we plan to integrate simulators that can mimic phosphoramidite synthesis and short-read sequencing with the OA-DSM pipeline. This way, we will generate an end-to-end simulated DNA archival system and demonstrate its integration with GLACIATION.

[24] Marinelli et al., Towards Migration-Free “Just-In-Case” Data Archival for Future Cloud Data Lakes, PVLDB 2023
[25] Yan et al., Scaling logical density of DNA storage with enzymatically ligated composite motifs, Nature Scientific Reports, 2023

6. Data management Services

In this section, we introduce a set of services that are used for metadata and data management in the GLACIATION platform. This includes six microservices for operating the DKG such as data storage service, metadata service, replica service, trade-off service, data processing and monitoring service, and prediction service, followed by secure data management service.

6.1. Data Storage Service

Technical description

The data storage service stores the data access patterns in the metadata service (see Section 6.2) to track data popularity within the platform. In addition, it writes to DKG with daily statistics on data access patterns and allows retrieval of daily summary statistics. Additionally, the microservice stores time series metrics and forecasting results from the prediction service (see Section 6.6), including aggregated daily energy consumption metrics and forecasting results from the prediction microservice of the DKG. Furthermore, it records the most recent history and predictions of energy consumption of the platform in the DKG. Beyond its storage functionality, the data storage service enables other services to retrieve the stored data access patterns and prediction history from the prediction microservice.

Technology (Internal Architecture)

The data storage service is a microservice developed in the Python programming language. It utilizes a time series database, such as InfluxDB[26], to store data access patterns from the metadata service and prediction service. The microservice is deployed as a container within the GLACIATION platform.

Interface and Interactions

The data storage service provides Python REST APIs using Flask[27] and Connexion[28] following the OpenAPI specification[29], which enables other services to write and retrieve data stored in the time series database. For instance, the service allows the metadata service to store data access patterns, maintaining records of data popularity. Similarly, it allows the prediction service to track energy consumption forecasting results in the database. Other services can query data access patterns or forecasting history from the prediction services via the REST API of the data storage service. Additionally, the service runs a batch job to update the last *n*-day data access patterns summary statistics and prediction results to DKG which can be accessed by other components in the platform such as the user interface of DKG, at the same time, ensuring that the data points written to DKG do not accumulate over time.

[26] Available at the following link: https://www.influxdata.com/
[27] Available at the following link: https://flask.palletsprojects.com/en/3.0.x/
[28] Available at the following link: https://connexion.readthedocs.io/en/latest/
[29] Available at the following link: https://spec.openapis.org/oas/v3.1.0

6.2. Metadata Service

Technical description

The component provides access to the DKG to all other services and components and allows to read information from it and edit information in it. The Metadata service communicates with all other service *via* HTTP protocol. It provides support for standard CRUD requests. On the other side the Metadata service queries local DKG using SPARQL query language.

For the *get* request the Metadata service uses swarm optimized search algorithm. For *post, put, delete* requests direct interaction with DKG is considered. However, the swarming algorithm can also be used for these tasks to a different extent. The necessity of the swarm algorithm usage for each case is currently under assessment.

Technology (Internal Architecture)

The Metadata service provides connection between HTTP requests and SPARQL queries of the database. The DKG partition which is accessed by the Metadata service is a Jena Fuseki instance. The processing of a CRUD request is schematically shown in Figure 30.

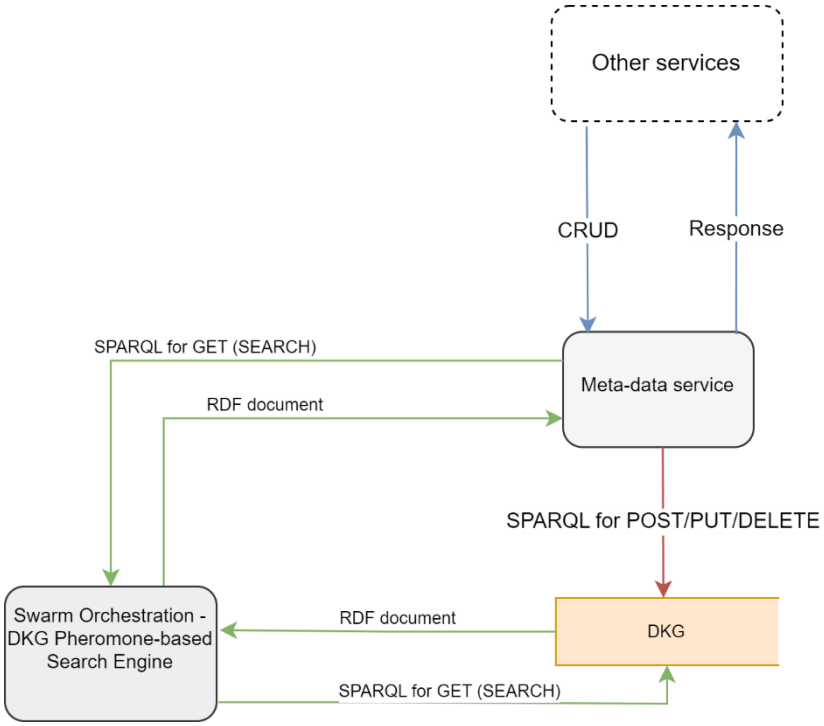

Figure 30: Schematic representation of the integrations of the Metadata service on one hand with all other services and with DKG on the other hand.

Interface and Interactions

Metadata service provides an HTTP interface for all other services to interact with the DKG. This is illustrated in the Figure 31. Metadata service interacts with all other services via HTTP protocol. It interacts with DKG using SPARQL queries. The search over DKG is facilitated by

swarm optimized search engine. For a get request Metadata service sends SPARQL query to the search engine and then the search engine sends back found RDF documents.

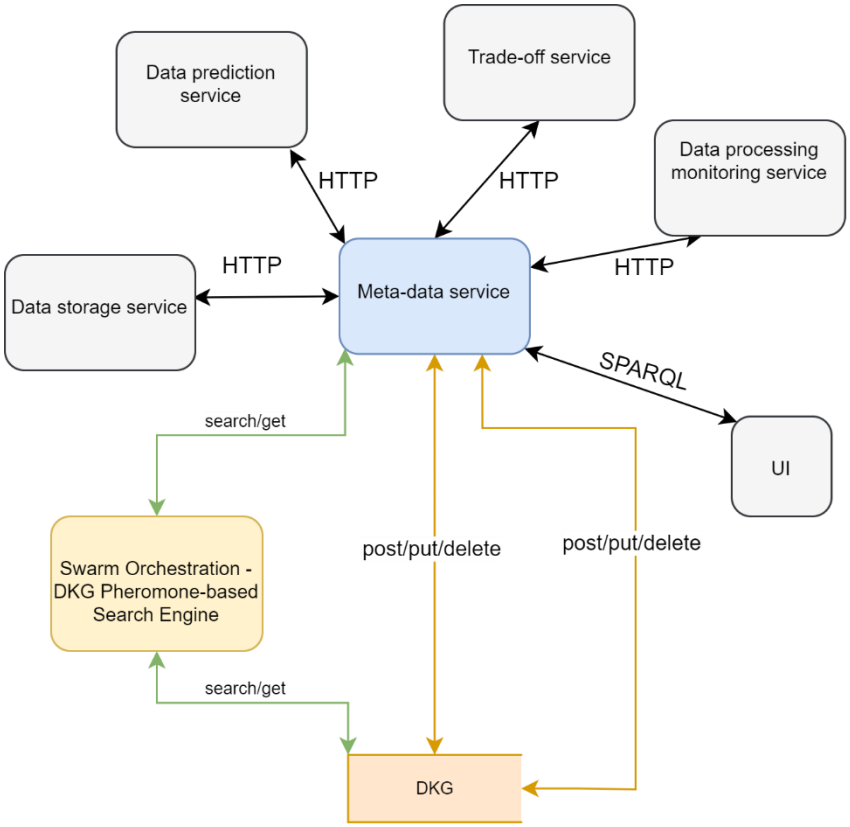

Figure 31: Metadata service (blue) interaction with other services (grey) and with DKG (red) with the help of the Search Engine (yellow). Interactions occur via either HTTP protocol or SPARQL queries.

6.3. Replica Service

Technical description

The purpose of this component is to ensure that a workload data is replicated to sustain the failure of a K8s worker node where the replica of the data resides. In case if this happens K8s scheduler will forward the calls to the other replica.

For AI/ML and Analytics workloads that process large data sets an object storage fits the best. The requirement for the storage is to have a possibility of replicating the data in the same K8s cluster as well as across K8s clusters in a simple way without much overhead.

Technology (Internal Architecture)

The technology proposed is MinIO object storage. The minIO is a popular storage system for AI/ML, distributed query, analytics, data lake components. The minIO is simple to use and can be installed in K8s cluster running on hardware with commodity JBOD disks (Just a Bunch Of Disks). The disk volumes can be encrypted with external key management system.

The replication can be done on the level of the cluster or site.

The replication on the level of the cluster is implemented with erasure coding. Minimal number of server nodes for distributed deployment is 4 and MinIO can tolerate the loss of up to a half the drives or nodes in the deployment while maintaining read access (“read quorum”) to objects. More details about erasure coding implementation can be found here[30].

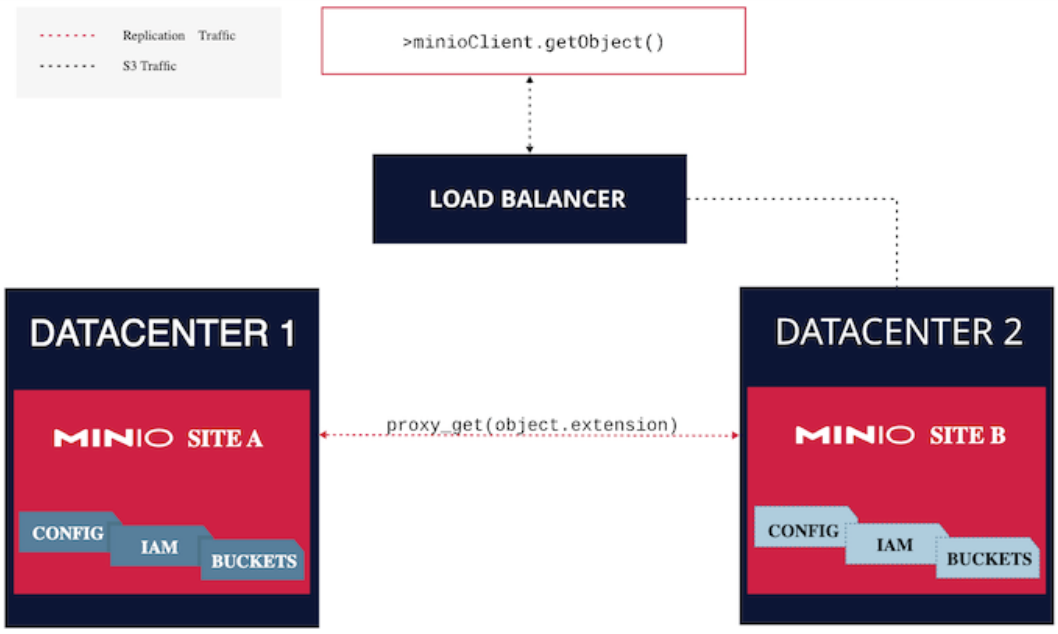

Figure 32: MinIO multisite replication.

The multisite replication is a form of replication between 3 or more MinIO deployments. This can be configured on the level of the bucket using rules. More details about server-side bucket replication are available here[31].

Interface and Interactions

The workload can read and write data using s3-compatible protocol. There are many libraries that can be used for accessing minIO buckets by applications[32].

6.4. Trade-off Service

Technical description

The Trade-Off Service is the bridge between the AI Decision Engine and InfluxDB. Whenever the AI Decision Engine needs to retrieve the current metrics of the system in terms of CPU, RAM, network, disk usage and energy, it can interact with the Trade-Off Service by invoking its APIs. In this way, the AI Decision Engine can send any kind of query, performing all the aggregations or filtering operations the different scenarios require. The Trade-Off Service receives the request with the required details, processes it and forwards it to the InfluxDB

[30] Available at the following link: https://min.io/docs/minio/linux/operations/concepts/erasure-coding.html#minio-erasure-coding
[31] Available at the following link: https://min.io/docs/minio/linux/administration/bucket-replication.html#minio-bucket-replication-serverside
[32] Available at the following link: https://min.io/docs/minio/linux/developers/minio-drivers.html?ref=docs

RESTful interface exposed by the query endpoint it is provided with. Once the query is solved, the database returns the result in the resolved response received by the Trade-Off Service, which normalises it and sends it to the AI Decision Engine.

Technology (Internal Architecture)

The Trade-Off Service is a microservice developed in Java and empowered by the Spring Framework to provide a flexible and scalable application.

Interface and Interactions

The component exposes a REST interface that enables interaction through some APIs that allow to perform advanced queries to the metrics data storage.

A high-level view of the Trade-off Service is shown in Figure 33.

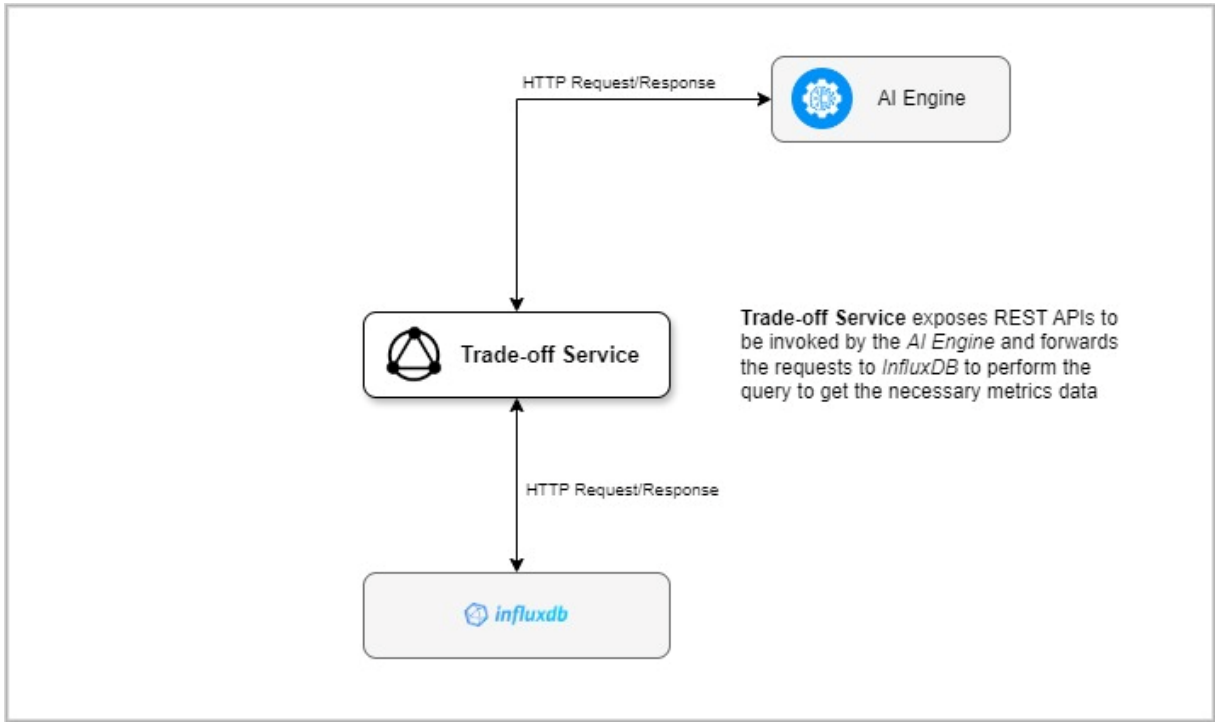

Figure 33: Trade-off Service High Level View.

6.5. Data-Processing monitoring Service

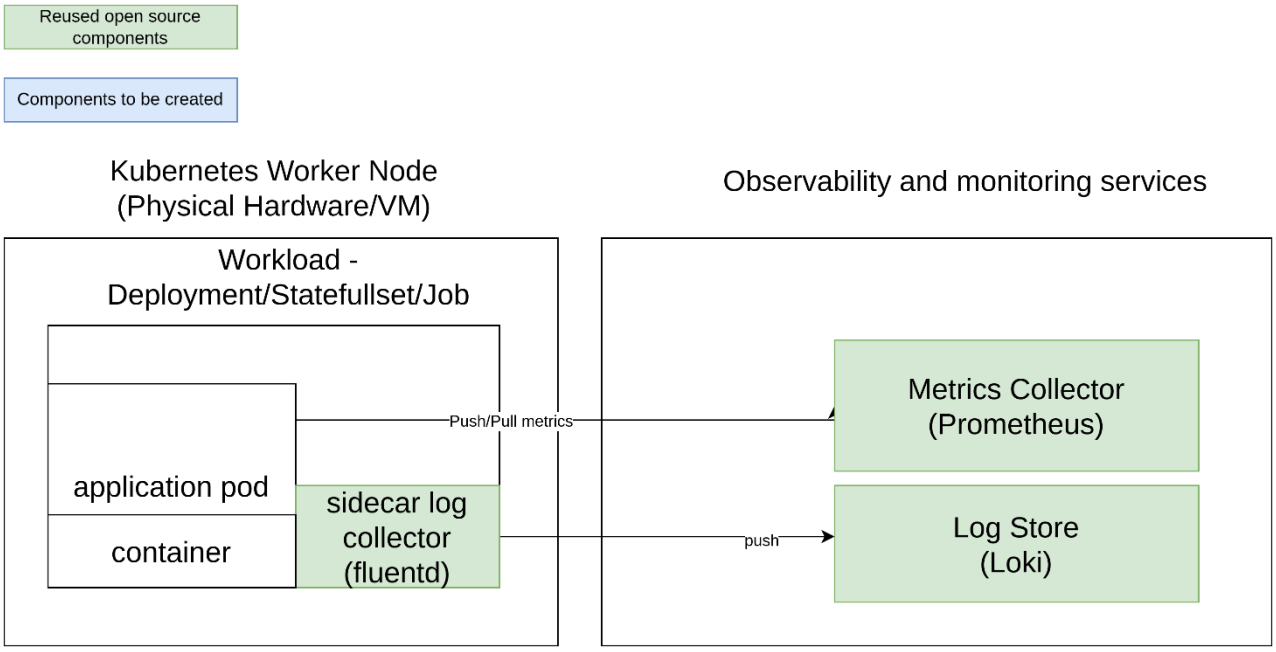

Figure 34: Data-Processing monitoring Service.

Technical description

The data processing monitoring service is the one responsible for monitoring the telemetry for the workload, adding them to a metrics collector and getting the container logs from the sidecar log collector, and storing them to Log Store.

Technology (Internal Architecture)

The following opensource components will be used for workload monitoring.

- Sidecar log collector (fluentd) - sidecar pod that collects the logs from the user workload and send these logs to log store.
- Metrics Collector (prometheus) - Metrics collector configured to scrape /metrics endpoint for all pods from all namespaces and to load metrics to the Metric Store.
- Log Store (Loki server) - Log store is used to aggregate logs.

Interface and Interactions

- Prometheus scrape – Metrics collector is a prometheus instance configured to scrape /metrics endpoint for all pods from all namespaces and to load metrics to metrics store.
- Metrics push - OpenTelemetry framework is used to actively "push" the collected metrics.
- Log Store – Log storage is an instance of Loki server. Together with fluentd side car and fluentd client they enable log aggregation in log store.

6.6. Prediction Service

Technical description

The prediction service provides forecasting functionality for time series data with a coarse time granularity on the platform, such as daily summary statistics of energy consumption. It retrieves raw energy consumption time series data from the interface provided by the performance measurement framework (Task 5.1) using a daily batch job to aggregate daily energy consumption statistics and forecasts energy consumption in the next day(s). The resulting forecasts are stored in the data storage service. Additionally, the prediction service allows other services to retrieve the most recent forecasting results.

Technology (Internal Architecture)

The prediction service is a microservice developed in the Python programming language, and it utilizes relevant packages such as pmdarima[33], Tensorflow[34] and Keras[35]. Similar to other microservices, the prediction service is deployed as a container within the GLACIATION platform. It offers forecasting methods, including simple yet efficient ones like AutoARIMA (AutoRegressive Integrated Moving Average) and more sophisticated ones using deep learning approaches developed in the work package 3 of the project.

Interface and Interactions

The prediction service, similar to the data storage service, provides Python REST APIs using Flask and Connexion following the OpenAPI specification. To retrieve the raw time series data from the performance measurement framework (Task 5.1), the prediction service first obtains metadata information from the metadata service. It then interacts with the performance measurement framework APIs to retrieve the raw data using a batch job. The raw time series data is pre-processed to a coarser time granularity for forecasting daily trends. The resulting daily prediction results are stored in the time series database used by the data storage service via its REST API. Additionally, the prediction service offers a REST API to allow other services to retrieve the most recent prediction results from DKG through the metadata service.

6.7. Secure data management Service

Technical description

Data management encompasses the systematic collection, organization, protection, and storage of an organization's data assets to facilitate informed business decision-making. It employs diverse techniques in data processing, storage, governance, and security, ensuring data quality, availability, and integrity. Effective data management is crucial for unlocking data-driven insights, enhancing operational efficiency, and adhering to regulatory requirements. The Secure data management service includes data lifecycle management (availability, destruction, access control, archival), optimized data storage across distributed edge

[33] Available at the following link: https://pypi.org/project/pmdarima/
[34] Available at the following link: https://www.tensorflow.org/
[35] Available at the following link: https://keras.io/

resources, and data provenance mechanisms for trust and quality. Additionally, the task will develop data sovereignty solutions, appropriate vocabularies, and strategies to mitigate microarchitecture-induced security vulnerabilities.

Technology (Internal Architecture)

The technologies used to implement the secure data management service are based on software from the SODA foundation[36]. They are broadly broken into two categories: storage and monitoring, and multi cloud data solutions. Terra is an open-source project focused on storage automation and management. It streamlines storage provisioning, data protection, and empowers self-service for users, reducing operational expenses. Terra offers a standardized API and supports diverse storage vendors, integrating seamlessly with platforms like K8s, OpenStack, and VMware. Complementary to Terra, Delfin provides open-source storage monitoring and alerting. It monitors performance metrics, uses predictive analytics to identify issues, and supports automated remediation across various storage systems, including EMC, IBM, NetApp, and more. This suite of open-source tools tackles multi cloud data challenges. Strato offers cloud-agnostic data management, enabling cloud backup, cloud switching, and content distribution for hybrid or multi cloud setups. Kahu focuses on K8s, providing backup, restore, disaster recovery, and migration capabilities for clusters and persistent volumes. Finally, Como acts as a virtual data lake, creating a centralized data repository with a unified interface, simplifying data aggregation, consolidation, and observability across multiple public and private clouds.

Interface and Interactions

Terra and Delfin provide RESTful APIs for streamlined storage provisioning, data protection, and monitoring across diverse storage vendors, integrating with platforms like K8s, OpenStack, and VMware. Terra automates storage management tasks, while Delfin focuses on storage monitoring, leveraging predictive analytics for proactive issue resolution. Strato facilitates cloud-agnostic data management, enabling efficient data operations across multiple clouds without the need to interact with complex cloud-native APIs directly. Kahu enhances K8s environments with data protection capabilities, using K8s-native constructs for backup and disaster recovery operations. Como serves as a virtual data lake, offering a unified interface for data aggregation and analysis, simplifying multi cloud data visibility. Additionally, it is expected a data space connector will be integrated into one of those tools.

[36] Available at the following link: https://www.sodafoundation.io/

7. Observability Services

In this section, we introduce some services for collecting telemetry metrics from different exporters, storing, and displaying them in dashboards, setting alerts, and collecting and storing logs in the GLACIATION platform. This includes two services telemetry and monitoring service and logging service.

7.1. Telemetry and monitoring

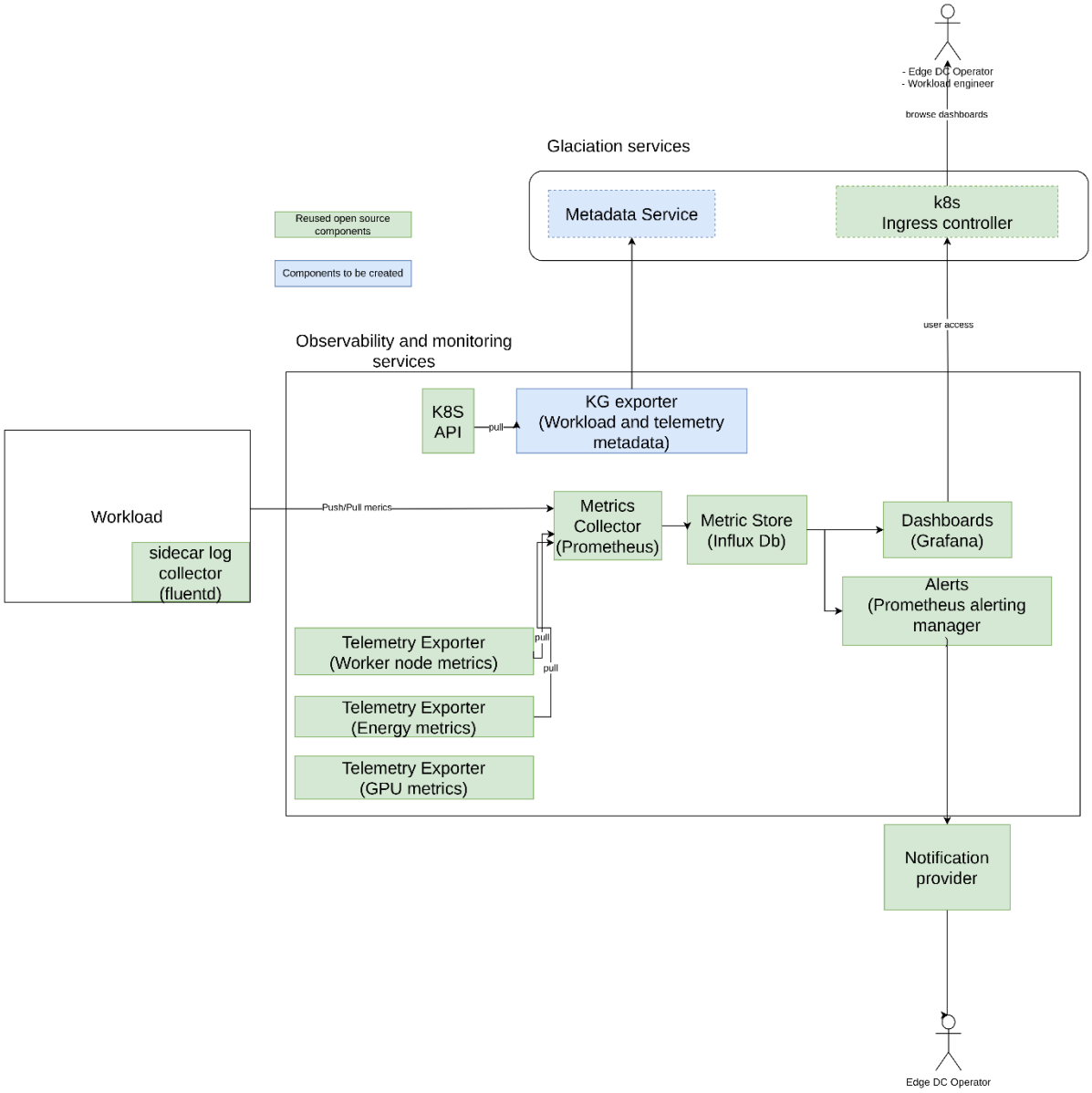

Figure 35:Telemetry and Monitoring services.

Technical description

The telemetry and monitoring service is the component responsible for collecting all the metrics from the worker nodes, Hardware power interfaces/NVidia DCGM and storing them in the Metric Store. Additionally, the framework collects metadata information about nodes and updates corresponding knowledge graphs - workload and workload telemetry in metadata service using KG exporter component.

The metrics stored in metric store then they can be used to create dashboards for visualization and alerts.

Technology (Internal Architecture)

The components of the Telemetry and monitoring services are given below:

- Telemetry Exporter (node metrics) - The daemon set that exports K8s worker node metrics.
- Telemetry Exporter (energy metrics) - Exports energy metrics for node, namespace, and workload scopes.
- Telemetry Exporter (GPU metrics) - Exports telemetry metrics for NVidia GPUs.
- KG exporter - this is a python service that listens to k8s API workload and update the corresponding knowledge graph - workload and workload telemetry in metadata service.
- Metrics Collector (Prometheus) - Metrics collector configured to scrape /metrics endpoint for all pods from all namespaces and to load metrics to metrics store.
- Metrics Store (Influx DB) - Metrics store timeseries database.
- Alerts (Prometheus alerting manager) - Alert manager integrated with notification provider.
- Dashboards (Grafana instance with Metrics Store data sources) – Dashboard for Log Store and Metrics Store data sources, some preconfigured dashboards with the main metrics.

Interface and Interactions

- Prometheus scrape – Metrics collector is a prometheus instance configured to scrape /metrics endpoint for all pods from all namespaces and to load metrics to metrics store.
- Metrics push - OpenTelemetry framework is used to actively push the collected metrics.
- Kg_exporter- inserts knowledge graph into metadata service using REST API.

7.2. Logging

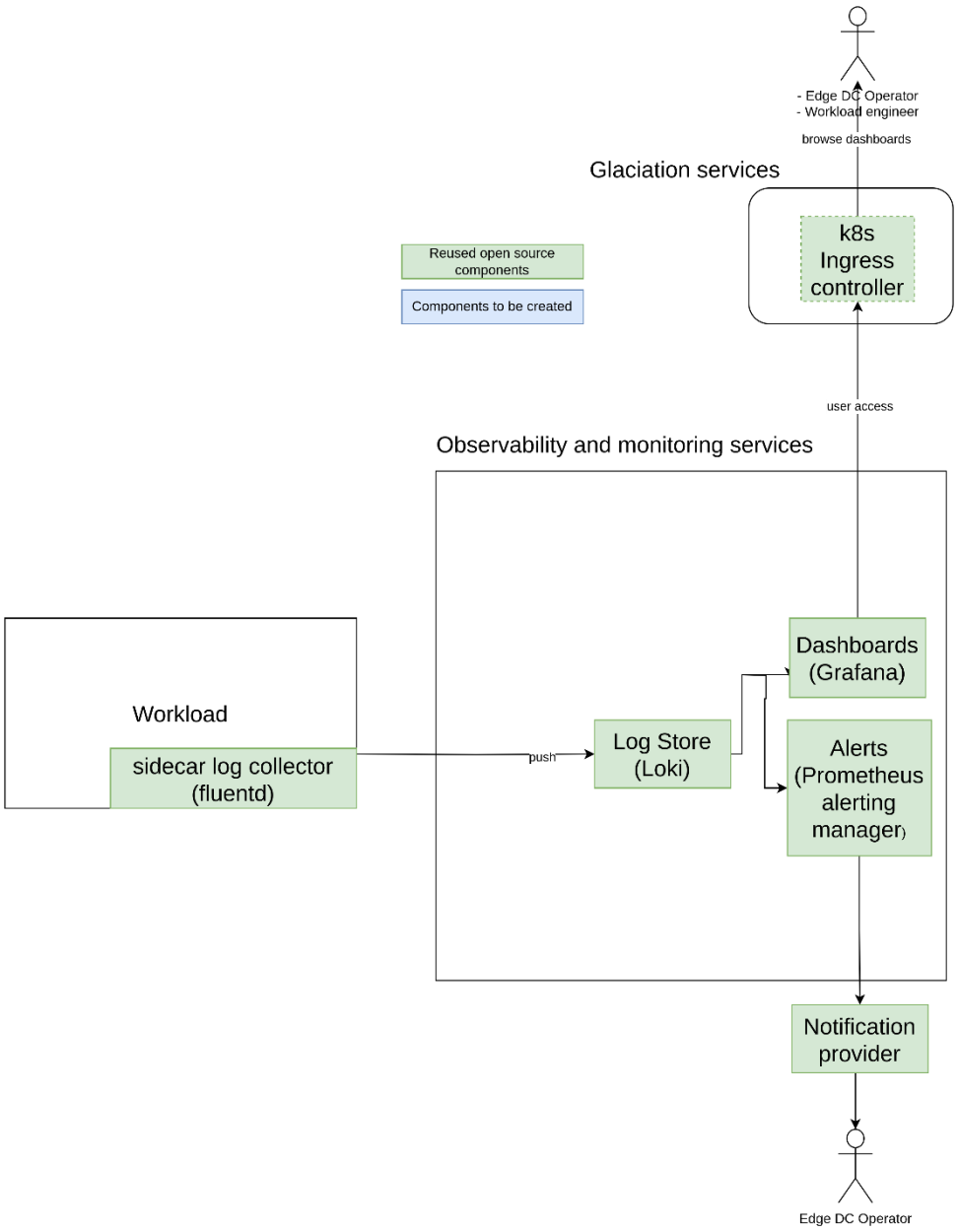

Figure 36: Logging services.

Technical description

Logging service is responsible for collecting the logs from the running containers and storing them so they can be used to create dashboards for visualization and alerts.

Technology (Internal Architecture)

- Alerts (Prometheus alerting manager) - Alert manager integrated with notification provider.
- Log Store (Loki)- Log storage is used to aggregate the logs in log store.

- Dashboards (Grafana instance with Metrics Store data sources) – Dashboard for Log Store and Metrics Store data sources, some preconfigured dashboards with the main metrics.

Interface and Interactions

- Logs – Log storage is an instance of Loki server. Together with fluentd side car and fluent client, they enable log aggregation in log store.

8. Management Services

The Management Services are covered in detail in this chapter. They are the platform's organizers, ensuring that everything functions properly. They facilitate seamless communication between various system components, enable integration, control, and manage all our resources as a single, cohesive unit, and automatically identify and register various services offered by the various components.

8.1. Device Discovery and cataloguing

Device discovery and cataloguing is the ability to identify, locate, and document all the devices present in the environment. This process involves creating a structured repository or catalogue that stores detailed information about each discovered devices, servers, deployed software and more. Device Discovery and Cataloguing helps detect unauthorized devices and ensures that only authorized devices are allowed to connect, administrators can monitor device health, performance, and troubleshoot issues more effectively, efficient resource allocation, configuration management, and maintenance tasks, assists in tracking and managing IT assets, automate workflows that involve provisioning, configuration, and software updates.

For instance, when utilizing the Dell Device Discovery Utility, it can discover a specific Dell device using either the SNMP or WS-MAN protocol, but not both simultaneously. Additionally, it can discover devices using any of the following methods:

- Device's IP address or FQDN.
- Subnet with mask.
- File containing a list of device IP addresses or FQDNs.

An illustrative example of new SNMP Discovery Source in the Figure 37 below.

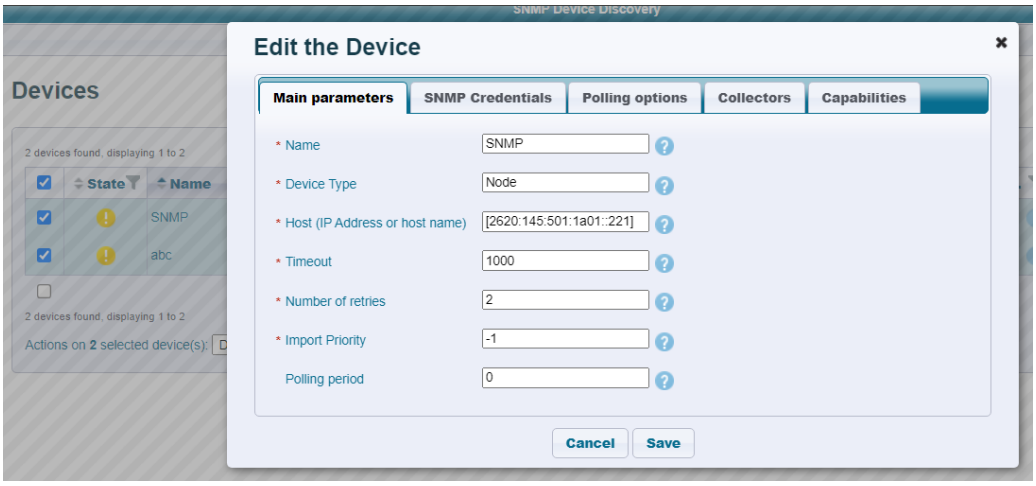

Figure 37: SNMP Device Discovery.

8.2. Service Discovery and cataloguing

Service discovery refers to the process of discover, track, and monitor the health of services within a network. Service discovery registers and maintains a record of all your services in a service catalogue. This service catalogue allows services to query and communicate with each other.

Service discovery and cataloguing concepts enable efficient communication, load balancing, and resource allocation within complex architectures where multiple services need to communicate and interact with each other, enabling seamless communication and interaction among services while promoting scalability, reliability, and agility.

Service discovery tools help microservices locate and communicate with each other as they dynamically scale, making them an essential element of a microservice architecture, providing benefits ranging from simplified scalability to improved application resiliency. Can utilize service discovery mechanisms like Consul or K8s' built-in service discovery in combination with load balancers.

Service discovery systems can be implemented across on-premises or cloud infrastructure. Additionally, service discovery is a native feature of container orchestrators such as K8s and Nomad and can be based on DNS or K8s API server.

An example of a Eureka Service Discovery Model is as shown in the Figure 38 below, Eureka is a 'mid-tier load balancer' built by Netflix and released as open source. It is designed to allow services to be able to register with a Eureka server and then locate each other via that server.

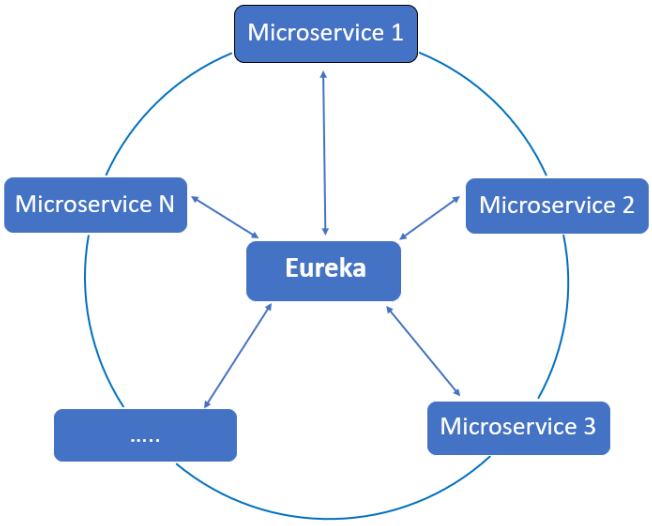

Figure 38: Eureka Service Discovery Model.

8.3. Cluster and federation forming

Cluster and Federation Forming introduces a centralised governance model where a single authority oversees the collective behaviour of the entire physical machines. By consolidating control, the system becomes adept at adapting to changing workloads. Multiple servers are so aggregated into a shared pool. This pooling mechanism enables load balancing mechanisms to distribute tasks across nodes and prevent bottlenecks.

The orchestration of several independent machines is possible thanks to the activities applied and described here. To achieve a unified and cohesive cluster, it's necessary to create a K8s cluster and add new machines to it. The K8s tool kubeadm is often used for this purpose. Its role goes beyond simply adding nodes; it orchestrates the process of configuring, initialising, and joining a new node to the existing cluster.

In a K8s cluster, there are two main types of nodes: Control Nodes and Worker Nodes. The Control Node acts as the central point for making decisions about the cluster's state and orchestrating the deployment of applications. It contains various native components like the K8s API server, which exposes the K8s API, Controller Manager, responsible for regulating controllers that manage the ideal state and the etcd data store, storing the configuration data of the cluster. Worker Nodes are where the actual workloads run. They execute tasks assigned by the Control Node and host the containers with the specific applications. Worker node components are a Kubelet, responsible for communication with the Control Node and managing containers on the node, a Container Runtime responsible for pulling and running container images and a Kube Proxy to maintain network rules and enable communication across the cluster.

Now that is clear the benefits of having a cluster formed in K8s and its description, adding a new Worker Node to a K8s cluster involves a series of steps:

1. Preparation of the New Node: installation of the necessary Container Runtime (e.g. Docker, containerd, generally depends on preference or k8s version used) and enabling network connectivity with the existing cluster.
2. Installation of K8s tools: installation of the Kubelet to enable communication with the Control Node.
3. Generate Token on Control Node: generate a token using the kubeadm token create command.
4. Join the Cluster: on the new node run the kubeadm join command with the token. This command initiates the joining process and establishes communication with the Control Node.

As seen here the step is clear and simple, as output the new node will contributes to the overall capacity and capability of the system.

9. Conclusions

In conclusion, Deliverable D2.1 represents a significant achievement for the GLACIATION project, providing a strong foundation through its architectural blueprint and service component definitions. Looking forward, our attention will be directed towards implementing and integrating the GLACIATION kernel functions.

Following the completion of Deliverable D2.1, the project will transition into the development phase, focusing on the creation of essential kernel functions such as device and service discovery, cluster formation, and telemetry infrastructure. This phase will demand meticulous attention to detail and adherence to the architectural blueprint to ensure the successful realization of project objectives.

Additionally, integration efforts will be undertaken to seamlessly integrate the GLACIATION kernel with other system components, including devices, clusters, clouds, and remote services. Collaboration with relevant work packages will be essential to facilitate smooth integration processes and ensure cohesive operation.

Moving forward, it is imperative to maintain a focus on modularity, scalability, and alignment with architectural requirements. By effectively executing these next steps, we can advance towards the successful development and integration of the GLACIATION platform, bringing us closer to achieving our project goals.