NEARDATA

Funded by the European Union

HORIZON EUROPE FRAMEWORK PROGRAMME

NEARDATA

(grant agreement No 101092644)

Extreme Near-Data Processing Platform

D2.2 NEARDATA Architecture Specs and Early Prototypes

Due date of deliverable: 30-04-2024
Actual submission date: 30-04-2024

Start date of project: 01-01-2023 Duration: 36 months

Summary of the document

Document Type	Report
Dissemination level	Public
State	v1.0
Number of pages	49
WP/Task related to this document	WP2 / T2.1, T2.2, T2.3, T2.4
WP/Task responsible	Universitat Rovira i Virgili (URV)
Leader	Xavier Roca (URV)
Technical Manager	Andre Martin (TUD), Aaron Call (BSC)
Quality Manager	Paolo Ribeca (UKHS), André Miguel (SCO)
Author(s)	Xavier Roca (URV), Raúl Gracia (DELL), Sean Ahearne (DELL), Aaron Call (BSC), André Miguel (SCO), Enrique Chirivella (KIO), Theodore Alexandrov (EMBL)
Partner(s) Contributing	URV, DELL, BSC, SCO, KIO, EMBL
Document ID	NEARDATA_D2.2_Public.pdf
Abstract	Specification of the Architecture and APIs. Documentation, early tutorials, and automated tests for the early prototypes of the different software components.
Keywords	Data space, extreme data, architecture, software components, component integration, confidential computing.

History of changes

Version	Date	Author	Summary of changes
0.1	10-03-2024	Xavier Roca (URV)	First draft.
0.2	12-04-2024	Xavier Roca (URV), Pedro Garcia Lopez (URV), Raúl Gracia (DELL), Sean Ahearne (DELL), Aaron Call (BSC), André Miguel (SCO), Enrique Chirivella (KIO), Theodore Alexandrov (EMBL)	Contributions.
0.5	15-04-2024	Andre Martin (TUD), Aaron Call (BSC), Paolo Ribeca (UKHS), André Miguel (SCO)	Internal review of the deliverable
1.0	30-04-2024	Xavier Roca (URV), Pedro Garcia Lopez (URV)	Final version.

Table of Contents

List of Abbreviations and Acronyms

AI	Artificial Intelligence
API	Application Programming Interface
AWS	Amazon Web Services
BSC	Barcelona Supercomputing Center
CA	Certification Authority
CAS	Configuration and Attestation Service
CNS	National Supercomputing Centre
CPU	Central Processing Unit
CSV	Comma-separated values
ECS	Amazon Elastic Container Service
EMBL	European Molecular Biology Laboratory
ETL	Extract, Transform and Load
GB	Gigabyte
GPU	Graphics Processing Unit
HPC	High-Performance Computing
HTTP	Hypertext Transfer Protocol
IoT	Internet of Things
IP	Internet Protocol
JSON	JavaScript Object Notation
K8s	Kubernetes
KPI	Key-Performance Indicators
LIDAR	Light Detection and Ranging o Laser Imaging Detection and Ranging
MDR	Multi Dimensionality Reduction
MS	Mass Spectrometry
MSI	Mass Spectrometry Imaging
PKI	Public Key Infrastructure
PoC	Proof of Concept
RAM	Random-Access Memory
SCO	Scontain
SDP	Streaming Data Platform
SGX	Software Guard Extensions

SLA	Service Level Agreements
SSD	Solid State Drive
SVR	Support Vector Regression
TCB	Trusted Computing Base
TEE	Trusted Execution Environment
TIFF	Tagged Image File Format
TLS	Transport Layer Security
UI	User Interface
UKHS	UK Health Security Agency
URL	Uniform Resource Locator
URV	Universitat Rovira i Virgili
VCF	Variant Call Format
VM	Virtual Machine

1 Executive summary

Deliverable D2.2 NEARDATA Architecture Specs and Early Prototypes aims to present the NEARDATA architecture specifications, show the integrations between components together with an early prototype and introduce the three NEARDATA International Health Data Spaces (Metabolomics, Genomics and Surgomics).

First of all, we present the NEARDATA architecture based on four modules: Analytics, Control Plane, Data Plane and Data Sources that incorporate the different components that make up the NEARDATA platform. In this deliverable, these components are briefly described along with the introduction of components developed from the theoretical definitions previously made in the previous deliverable *D2.1 Initial Architecture Specifications*. Next, we introduce the resources needed to deploy and run the different components on the NEARDATA testbed. Additionally, we provide a description of the component integrations and show the first early prototypes that have been developed during the project. Finally, we present the three NEARDATA International Health Data Spaces in the different fields of metabolomics, genomics and surgery together with their respective KPIs to validate them.

2 Introduction

According to multiple analyst sources, between 80 and 90% of existing data are unstructured, in text, images, audio, video, among others. This makes it difficult to store and search them in predefined schemas established in traditional databases. As a consequence, huge amounts of unstructured data are stored in object stores, which adds great complexity to their extraction and analysis. This type of massive and complex unstructured data can be defined as Extreme data. The management of extreme data (discovery, collection, extraction, filtering and processing) remains difficult for data scientists, as they have to deal with a proliferation of data sources from heterogeneous domains. Each of these includes different domain-specific data manipulation mechanisms, data pipelines, and governance models with which data scientists must become familiar.

The project focuses on three health data domains containing large amounts of unstructured data: metabolomics (images), genomics (text) and surgical data (video). The data management and processing needs and characteristics of these domains fit perfectly into the definition of extreme data. The massive increase in the volume of OMICs data leads to a technological problem that pushes workstations to the limit. The current ingest-then-compute data-shipping paradigm [1] does not respond to the new needs originated by this problem. For this reason, a big challenge arises for data ingest from object stores to analytical computing platforms. Clearly, moving massive data into the memory of analytic clusters to be preprocessed and manipulated can become extremely expensive. This problem is magnified when the data is remotely located on the Edge due to added network latencies. In addition, computer-assisted surgery presents an extreme problem based on the speed of data processing, adding a strong requirement on providing low latency in real-time video analytics and robotic IoT event streams. For this reason, an extreme challenge arises to process real-time video streams over massive data stored in the object store. Thus, to cope with this problem, the near-data paradigm appears to process data at the edge that can meet the needs presented. Finally, dealing with highly sensitive health data implies demanding compliance with the security and privacy requirements provided by hospitals and laboratories. The big challenge here is to discover and distil meaningful, reliable and useful data from heterogeneous and dispersed/scarce sources in a trustworthy way with stringent confidential requirements.

The objective of the NEARDATA project is to design an extreme data processing platform inspired by the near-data paradigm. This concept can be described as the ability to set up computation and processing close to the data, thus avoiding the ingestion of non-preprocessed data directly on the analytic clusters. Thus, this platform enables the consumption, mining and processing of distributed and federated data without needing to master the logistics of data access across heterogeneous data locations and pools, moving away from traditional passive data ingestion on storage systems to introduce a new generation of near-data processing platforms in the cloud and in the edge.

To put it all together, our NEARDATA platform is a novel technology for data mining of large and sparse unstructured data sets that can be deployed in the Cloud and on the Edge (HPC, IoT Devices), which leverages advanced AI technologies and offers a novel layer of sensitive confidential cybersecurity for trusted data computation.

Based on the needs introduced, three core objectives can be established:

- **O-1 Provide high-performance near-data processing for Extreme Data Types:** The first objective is to create a novel intermediary data service (Data Plane / Xtreme DataHub) between Object Storage and Analytic platforms. This Data Access Layer will provide serverless type-aware data connectors that optimise data management operations (partitioning, filtering, transformation, aggregation) and interactive queries (search, discovery, matching, multi-object queries) to efficiently present data to analytics platforms. Our data connectors facilitate an elastic data-driven process-then-compute paradigm which significantly reduces data communication on

the data interconnect, ultimately resulting in higher overall data throughput.

- **O-2 Support real-time video streams** but also event streams that must be ingested and processed very fast to Object Storage: The second objective is to seamlessly combine streaming and batch data processing for analytics. To this end, we will develop stream data connectors deployed as stream operators offering very fast stateful computations over low-latency event and video streams. In particular, we want to explore how native stream serialisers for heterogeneous data types (OMICs) can optimise intensive data flows between Object Storage and analytics services across the entire Compute Continuum.
- **O-3 Provide secure data orchestration, transfer, processing and access:** The third objective is to create a Data Broker service enabling trustworthy data sharing and confidential orchestration of data pipelines across the Compute Continuum. In order to ensure confidentiality, integrity and freshness of the data, the project will develop mechanisms to utilise Trusted Execution Environments (TEEs) along federated learning architectures. To protect data in flight and rest, the project will develop mechanisms for transparent encryption for data transfers as well as storage that require no code modification and provide high throughput at the same time. Policy-driven mutual authentication mechanisms will furthermore support advanced data access policies for non-trusting stakeholders which are governed through policy boards so that access rules can be dynamically updated by human deciders and other entities.

The three objectives will be validated in real settings using the following Key-Performance Indicators (KPIs):

- **KPI-1** - Significant performance improvements (data throughput, data transfer reduction) in Extract-Transform-Load (ETL) phases validated with near-data connectors over extreme data volumes (genomics, metabolomics).
- **KPI-2** - Significant data speed improvements (throughput, latency) in real-time video analytics validated using stream data connectors.
- **KPI-3** - Demonstrated resource auto-scaling for batch and stream data processing validated thanks to data-driven orchestration of massive workflows.
- **KPI-4** - High levels of data security and confidential computing validated using TEEs and federated learning in adversarial security experiments.
- **KPI-5** - Demonstrated simplicity and productivity of the software platform validated with real user communities in International Health Data Spaces.

3 Ambition

The NEARDATA proposal is ambitious and goes beyond the state of the art in diverse aspects aligned with the objectives mentioned above. We distinguish between Background, Innovation and Research Ambition.

3.1 Background

In recent years, data sharing has driven the development of innovative technologies and solutions to complex problems that previously presented extremely difficult challenges due to the privatization of data and consequently, the lack of transparency of effective data processing solutions. By sharing data, organizations and researchers benefit from access to third-party data to enrich research in different fields. To this end, participants who exchange data must ensure that data communication is established securely and with confidence through governance mechanisms and policies.

Data Spaces emerge as a solution to facilitate data exchange between organizations and researchers. Data Spaces go beyond the simple exchange of data to implement an ecosystem capable of guaranteeing secure data sharing. For this purpose, different entities and mechanisms are established to guarantee secure data roles and access.

The Reference Architecture of International Data Spaces[1] has been the source of inspiration for the development of NEARDATA's architecture.

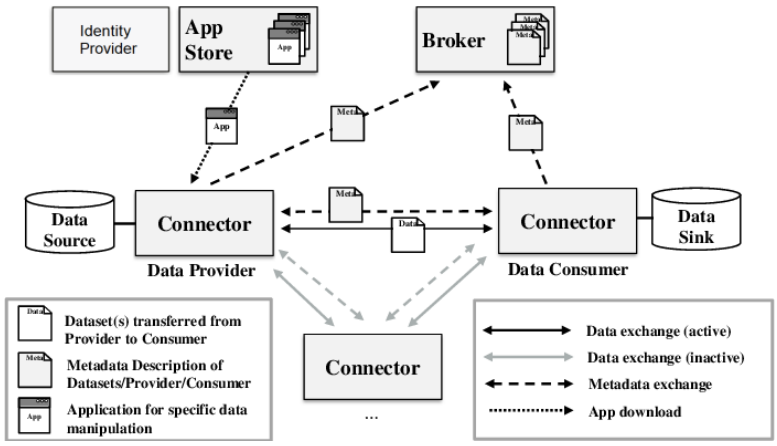

Figure 1: International Data Spaces Reference Architecture.

From Figure 1 we can identify the key entities that form the reference architecture. According to the International Data Spaces Association, entities can be defined as follows:

- The **Data Owner/Source** is an entity or a person capable of creating data or executing control over it. Likewise, Data Owner/Source can define policies to provide secure access to its data. In this way, the Data Owner/Source makes the data available for the Data Provider.
- The **Data Provider** is responsible for providing the data to the Data Consumer from the Data Owner. Additionally, the Data Provider facilitates data discovery by the Data Consumer by sending metadata about the stored data to the Data Broker.
- The **Data Consumer** is in charge of receiving the data from the Data Provider. The Data Consumer can perform data sourcing from the Data Broker.
- The **Data Broker** is an intermediary service to manage information about the data stored in Data Owners/Sources. The Data Broker allows to receive metadata from Data Providers to be

[1] `https://internationaldataspaces.org/`

queried by Data Consumers.

- The **Data Connector** is a key entity that enables secure and efficient data communication and sharing by using policies to ensure data sovereignty. It aims to connect data endpoints to augment the available datasets to turn Data Spaces into secure data environments to exchange data freely.
- The **App Store** aims to provide data applications deployed on Data Connectors. These data applications present different data processing workflows that are managed by the App Store.
- The **Identity Provider** manages user identities to perform data creation, management, monitoring and other operations. In addition, it controls the access of identities to the data in order to prevent the security and confidentiality of this.

3.2 Innovation

NEARDATA seeks to progress the state of the art by developing a novel Data Space architecture based on new definitions of the key entities of the International Data Space reference architecture. Entities such as Data Broker and Data Connectors form the core of NEARDATA's architecture, and for this reason, their concepts are redefined as innovations within our Data Space.

The Data Broker is responsible for ensuring secure access and discovery to data and the confidential execution of analytical applications in our platform. Its core is formed by two key components: the confidential data exchange, a rich metadata platform for data encryption to facilitate private data discovery, and a confidential and federated orchestration layer for edge workflows that fulfill security and confidentiality requirements. The NEARDATA Data Connector aims to establish optimized data management operations based on new Extract, Transform and Load techniques to efficiently handle extreme health data.

NEARDATA goes beyond the entities established in the reference architecture by developing an XtremeHub platform that aims to incorporate near-data high-performance serverless type-aware data connectors suitable for extreme unstructured data, combining batch with low-latency stream processing, while preserving security and privacy through secure AI-enabled orchestration and Trusted Executions Environments in a distributed and federated environments across the Compute Continuum.

Today, the DataHub concept can be defined as an extensible metadata platform that allows efficient search and query of data stored at a single point[2]. DataHubs also provide data analysis features such as data discovery and data observability. Services such as Linkedin's DataHubPlatform[2] and Lyft's Amundsen[3] are clear examples of popular open-source DataHub platforms. Extreme data, popularly known as massive unstructured data with size disparity and different types and formats, are totally forgotten in this type of solutions. The complexity to manage extreme data becomes the the key focus of research and analysis. In the context of NEARDATA, the research lines we present are related to health data research such as genomics, metabolomics and surgery.

The "ingest-then-compute" pattern traditionally used for data analysis is inefficient when processing extreme data. The nature of this type of data makes preprocessing or data partitioning an extremely complicated challenge. NEARDATA wants to address this problem by extending the DataHub concept to deal with extreme data. The novel XtremeHub will include reliable, high-performance data connectors that will allow data analytics platforms to access and discover extreme data stored in Object Storage. The addition of stream data connectors will enable the unification

[2] `https://datahubproject.io/`
[3] `https://www.amundsen.io/`

of batch and stream processing to efficiently process real-time IoT data streams and provide long-term stream retention for high-performance batch processing. In addition, the platform will be able to efficiently orchestrate different workflows and data analysis platforms from optimized AI-based solutions. Finally, to address sensitive health data requirements, our platform will offer TEE-based solutions to deploy secure and privacy-preserving data flows across the entire Compute Continuum.

3.3 Research

NEARDATA seeks to progress beyond the state of the art in three main topics: near-data computing, stream analytics, and confidential federated computing.

Near-data computing. The near-data paradigm has been one of the fields studied in depth by the scientific community due to the advantages of bringing computing closer to the data: data locality significantly reduces latency and consequently increases bandwidth due to the reduction of data to be transferred to the computing platform[1]. Active Storage[3] has received attention as it has demonstrated data transfer savings of up to 95% and up to five times faster execution time. However, this solution has several weaknesses: there is the possibility of storage node congestion and it precludes multi-tenant services in the Cloud. To prevent the disadvantages of Active Storage, the idea of component disaggregation arose. Disaggregating the compute component from the storage layer allows the advantages of near-data to continue to be exploited. Thus, we find a close relationship with the serverless computing paradigm, which allows a massive horizontal scaling of resources by adapting to compute needs with fine granularity[4][5]. In this line, a first approach from Amazon Web Services (AWS) are Object Lambdas[6], which allow to execute inline Lambda functions as part of the data stream in a synchronous request to S3. Once again, existing solutions lack the extreme data support to guarantee massive unstructured data flows. For this reason, there is an open challenge for the research and development of a near serverless data processing layer for extreme data.

Stream analytics. Edge applications based on the analysis and processing of telemetry, image and video streams require low-latency and real-time data processing. The rise of extreme data is creating a technically complex challenge for data stream processing. To cope with this problem, there are existing solutions such as the "Lambda Architecture"[7], which combines real-time data processing and batch analysis. Batch data processing achieves excellent performance, but is inefficient for low-latency, real-time analysis[8]. The investigation of unifying batch and stream processing became very important. For example, Apache Beam and Apache Flink, unify stream and batch analytics by distinguishing whether datasets are “bounded” (batch) or “unbounded” (stream). The lack of an interconnected processing platform with auto-scalable Stream Storage layer mechanisms opens a research opportunity to facilitate elastic deployment and execution of low-latency real-time data connectors.

Confidential federated computing. NEARDATA will ensure the confidentiality and integrity of applications and data by using state-of-the-art technologies such as the TEE provided by intel SGX. This technology, still in the process of widespread adoption, presents a drawback to be taken into account: it is limited to specific vendor and CPU models, such as Intel Skylake CPU, etc. Therefore, there is a need to investigate compatibility requirements with different CPU vendors with similar security features to enable service providers to run their applications in isolation with attestation[9]. The goal of NEARDATA is to develop a transparent framework for the underlying hardware that guarantees secure and confidential workflows and data access by combining methods for the reliability of federated identities with the protection of computing platforms offered by TEEs.

NEARDATA shows an extremely ambitious proposal. We intend to present a completely novel XtremeHub technology for efficient AI-based extreme workflow orchestration and confidential extreme data management and access through the integration of TEEs.

4 Architecture Specifications

4.1 Proposed Architecture

From the reference architecture introduced, Figure 2 depicts the proposed architecture based on the actual components presented by each of the NEARDATA partners. Our complex and rich extreme near-data processing platform features four modules to encapsulate the specific requirements and tasks of NEARDATA components: Data Plane / XtremeHub (See *D3.1 XtremeHub first release and documentation*), Control Plane (See *D4.1 Data Broker release and documentation*), Analytics and Data Sources. In addition, we find the different use cases that will leverage the NEARDATA platform (See *D5.1 First release of KPI benchmarks in all use cases and data connector libraries)*. In our NEARDATA platform we observe five entities of the reference architecture: Data Providers, Data Consumers, Data Connectors, Data Broker and Identity Provider.

To facilitate the understanding of the proposed architecture, we will make a brief summary of the four modules and their components. In this deliverable, we will detail in more depth the integration of the components that form NEARDATA. For a more extensive description of all the components that make up NEARDATA and its APIs, we redirect the reader to the deliverable *D2.1 Initial Architecture Specifications*.

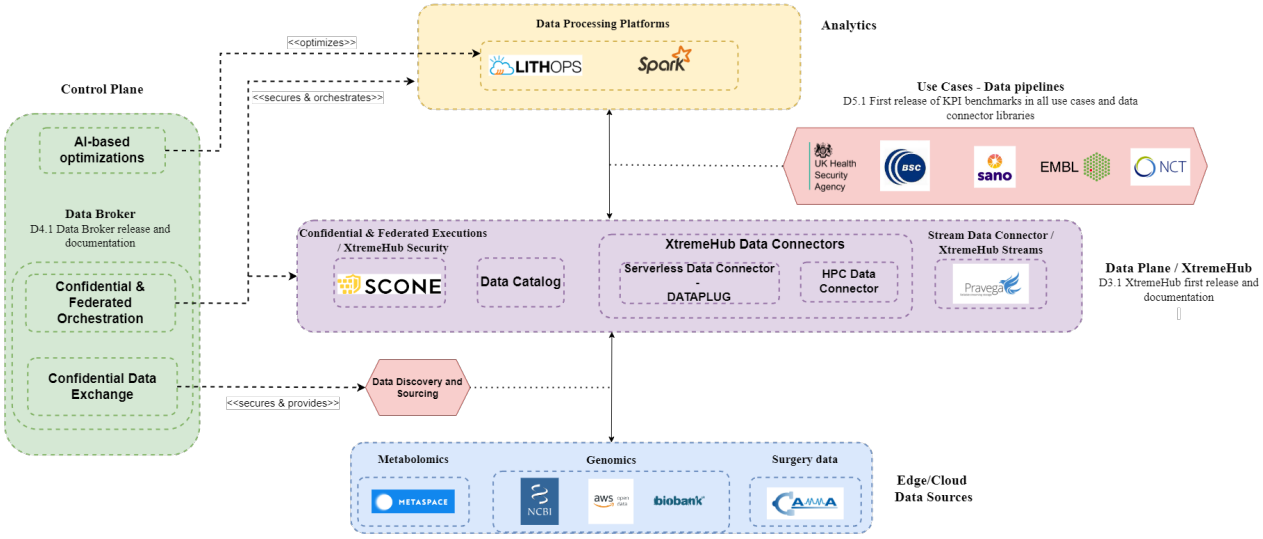

Figure 2: NEARDATA Architecture.

Our Software-Defined Data Infrastructure is divided in the Data Plane and the Control Plane. The Data Plane (XtremeHub and Data Connectors) is addressing objectives O-1 and O-2 validated through KPIs 1-2, while the Control Plane (Data Broker) is addressing objective O-3 validated through KPIs 4-5.

4.1.1 Analytics: Data Processing Platforms

Processing massive unstructured data is extremely computationally challenging, forcing data scientists to push the technologies they work with to the limit. Our extreme data infrastructure aims to mediate the data flows between Data Sources and Data Analytics platforms through our software components located in the Control Plane and Data Plane. The Analytics module incorporates Data Analytics platforms capable of processing extreme data from mechanisms that ensure large-scale computation.

Our expert partners in health big data computing present Edge or Cloud-based solutions across Data Processing platforms. To deal with each health data format, each partner presents its specific Data Pipeline as a solution. In relation to the International Data Spaces reference architecture, data scientists through their Data Pipelines across Data Processing platforms would be identified as Data

Consumers.

Apache Spark[4] is one of the most widely used data analytics platforms by data scientists capable of being deployed on multiple clusters adapting to the workload and scale of the data. Also, incorporating cloud services allows data scientists to deploy their own specific pipelines on virtual machines or cloud functions without the need to manage the data processing infrastructure. In our platform, the Lithops component makes it easy for data scientists to deploy workloads on cloud providers thanks to its multi-cloud support. Lithops[5] is a distributed computing framework for data analysis at massive scale that fits perfectly into highly parallelizable programs without the need for inter-process communication, but also supports parallel applications that need to share state between processes.

4.1.2 Data Sources: Edge/Cloud

The majority of data currently in existence is unstructured data. Due to their lack of structure or schema it is impossible to store them in traditional databases. As a result, it is common to find object storages filled with extreme amounts of unstructured data. In NEARDATA we present different massive sources of health data specific to each format.

In the field of genomics, we find the National Center for Biotechnology Information[6] and UK Biobank[7] that incorporate a large amount of secure and open biomedical and genomic data for research. In the field of surgery, Camma[8] is widely used for research and analysis of surgical videos. Finally, in our platform, we introduce the METASPACE[9] metabolomic Data Space as a direct component as Data Source for the field of metabolomics. The METASPACE platform hosts an engine for metabolite annotation of imaging mass spectrometry data as well as a metabolite spatial knowledge base of thousands of public datasets provided by the community.

The Data Sources are considered as Data Providers in the NEARDATA architecture as they securely enable data discovery, sourcing and consumption through their platforms.

4.1.3 Data Plane / XtremeHub: Data Connectors

Dealing with extreme data is a difficult challenge for data scientists. To this end, the NEARDATA Data Plane / XtremeHub platform module presents different tools that facilitate the ingestion and management of massive unstructured data. The Data Plane aims to convert bulk unstructured data blobs into semi-structured data sets for easy data management operations. In response, we introduce the novel concept of Extreme Data Types, virtual objects that encapsulate unstructured data and specific metadata. NEARDATA specialized Data Connectors will be able to manage and generate this data type to facilitate extreme data processing. In addition, the Data Plane incorporates containerized trust environments for secure data management and processing.

Within this module, five core components can be distinguished: Confidential & Federated Executions, Data Catalog, Serverless Data Connectors, Stream Operators and Data Programming abstractions (Analytics interconnection layer).

- The **Confidential & Federated Execution (XtremeHub Security)** is responsible for implementing security and confidentiality mechanisms to establish secure containerized environments to protect user applications by ensuring the processing of sensitive data. Later in the Control Plane, this type of environment will be orchestrated from the Confidential & Federated Orchestration layer found within the Data Broker. In our platform, the SCONE component allows to execute programs within Trusted Execution Environments with minor modifications to adapt the code to the component requirements. This guarantees secure executions through the identification of users that comply with the restrictions and policies established in the Data Broker.

[4] https://spark.apache.org/
[5] https://github.com/lithops-cloud/lithops
[6] https://www.ncbi.nlm.nih.gov/
[7] https://www.ukbiobank.ac.uk/
[8] http://camma.u-strasbg.fr/
[9] https://metaspace2020.eu/

- The **Data Catalog** stores metadata relevant to data preprocessing, partitioning, ETL tasks and data query mechanisms. The METASPACE Data Source for metabolomic data introduced above incorporates in its architecture a Data Catalog of its own. It facilitates the user the search and discovery of massive preprocessed data, thus complying with the near-data paradigm.
- The **Serverless Data Connector (XtremeHub Data Connectors)** allows to generate metadata and indices from extreme data to generate a virtual Data Catalog. It allows to adapt to the user's needs to dynamically transform and partition data. In this deliverable, we present the novel serverless data connector Dataplug described theoretically in deliverable *D2.1 Initial Architecture Specifications*. Additionally, Lithops can also be introduced as a Serverless Data Connector thanks to the incorporation of its Storage API that allows the connection between serverless data processing services and object storage.
- The **HPC Data Connectors (XtremeHub Data Connectors)** are developed to take advantage of high-performance computing (HPC) platforms that provide high-performance networking, compute and storage capabilities to handle extreme data efficiently and with low latency.
- The **Stream Data Connector (XtremeHub Streams)** service deploys connectors as auto-scalable stream operators that provide very fast stateful computations over low-latency video and event streams. Dell Pravega[10] is an open source storage system for data streams that translates the unified view of stream and batch analytics to the data storage via the Stream abstraction.
- The **Data Programming Abstractions** provide a layer of interconnection with Data Processing platforms (Data Consumers). At NEARDATA, we introduce Lithops as a computing framework for deploying massively parallel jobs in Edge and Cloud environments, thus ensuring the connection between the components that make up the Data Plane and the Analytics module.

4.1.4 Control Plane: Data Broker and Confidential Data Orchestration

The Control Plane is the major front-end of the NEARDATA platform which includes both data discovery, governance and acces but also optimised orchestration and declarative interconnection of heterogeneous data flows. Control Plane presents one of the most important and attractive features of our rich NEARDATA platform. It will be the hub for managing and controlling jobs and data access in a totally secure way. Sensitive health data requires highly restrictive handling requirements. Therefore, our Control Plane will guarantee secure executions with controlled data access.

The Control Plane will be designed to offer efficient executions through specific artificial intelligence mechanisms for data analysis platforms. This will be achieved through an intelligent and adaptive layer of interconnection between computing and storage dataflows thanks to the Data Plane. As a consequence, the Control Plane will guarantee the ability to consume data and run analysis jobs from anywhere efficiently and securely without the need to design and manage a complex infrastructure.

Four key components are included in this module:

- The **Confidential Data Exchange** will be in charge of securing the data coming from the Data Sources (Data Providers) to be confidentially consumed by the Data Consumers (Data Pipelines). Thanks to the mechanisms introduced in the Trusted Execution Environments it will allow to create, register and federate datasets of a specific domain, generating a rich metadata platform of encrypted datasets and favoring the discovery of confidential unstructured or semi-structured data.
- The **Confidential & Federated Orchestration layer** is a declarative interconnection framework for extreme data workflows ensuring confidentiality and security in the entire data path. The orchestration layer incorporates mechanisms for transparent encryption of data flows, and

[10] `https://cncf.pravega.io/`

leverages privacy-aware data connectors that conform to the data governance policies defined in the Data Broker. This layer manages the different components of the Data Plane to facilitate the confidential interconnection of data between the Analytics and the Data Sources modules.

- The **Data Broker** is the cockpit of the NEARDATA platform, exposing and orchestrating all services in the Data and Control planes. It encapsulates and orchestrates the two key components presented above. In our platform, SCONE will be in charge of orchestrating the deployed TEEs to ensure secure execution of the entire workflows. The Data Broker will incorporate a user authentication service for accessing and processing sensitive data that complies with the established restrictions and policies. TEEs have two components necessary to verify the confidentiality of applications and users. Configuration and Attestation Service (CAS) identifies when an application is trusted while KeyCloak manages user authentication and policies. Likewise, we identify our Identity Provider within our rich Data Broker.

- The **AI-based Optimiser** of Cloud/Edge Workflows is a learning service that focuses on improving data-driven orchestration of workloads and pipelines defined in the Orchestration layer. It will use state of the art deep and statistical learning methods to analyse data-bound complex workloads and optimise resource consumption and KPIs (performance) using telemetry information. It will generate efficient execution plans that meet policy constraints and instrument the orchestration layer. Our partner BSC will be in charge of providing this tool for the efficient orchestration on the Lithops computing framework.

4.2 Data Plane Components Recap

In this section, we briefly describe each of the components of the NEARDATA architecture that make up the Data Plane.

4.2.1 Lithops

Lithops is a serverless analytics data platform implemented in Python that enables the deployment of workflows on a massive scale across main cloud providers. Lithops is a perfect fit for heavy workloads that can be fully parallelizable without the need for inter-process communication.

Lithops offers an extensible storage and compute backend architecture fully agnostic to any cloud service, open source on-premise installations (Kubernetes, OpenShift) and Edge platforms (WebAssembly). These backends can be used by calling their respective APIs: the Compute API, which allows massively parallel tasks to be invoked on the selected compute backend, and the Storage API, to facilitate access to Object Storage. In addition, Lithops features an extensible on-the-fly partitioner architecture that allows parallel data consumption from the Object Storage. Lithops offers solutions for the partitioning of genomic and metabolomic data formats. The different Lithops execution modes allow to adapt to the needs of the user's serverless architecture, offering the possibility to deploy the processing platform locally, in virtual machines or in cloud functions.

4.2.2 Serverless Data Connector: Dataplug

In the deliverable *D2.1 Initial Architecture Specifications*, we theoretically propose the Serverless Data Connector component. Then, we will introduce the novel Dataplug framework developed from the specifications described in the aforementioned deliverable.

The seemingly "infinite" compute and storage resources available in the Cloud have paved the way for handling unprecedented amounts of unstructured data, marking the evolution from big data to what is termed as extreme data[10]. Extreme data is characterized by its increasing volume, speed, variety, complexity and extreme variations in values, which challenge current distributed computing technologies that struggle to cope with such demanding characteristics.

To analyze these extreme data repositories, researchers must first deal with data partitioning, which is crucial for efficient workload distribution[11] and to fully harness the scalability and parallelism of Cloud resources [12]. We understand data partitioning as splitting a large dataset into

smaller logical portions, so that each portion (or partition) can be processed by a distributed worker, thus increasing the scalability of data-parallel applications.

However, partitioning extreme unstructured data at this scale is challenging. A common approach for unstructured data is to partition a dataset into smaller even-sized files or objects, or transform it into a "Cloud-friendly" format[13]. This, however, involves reading, pre-processing, and writing back to storage the whole dataset, which can become extremely costly. Moreover, this issue is further exacerbated in Cloud scenarios, where object storage serves as the primary storage for scientific computing in the Cloud[13]. Since objects are immutable, partitioning a dataset into many objects requires rewriting all the data, making this approach inefficient. Finally, arbitrarily setting a fixed partition size without any hints can result in suboptimal performance[11], given the variation in workload requirements. Thus, we argue that static partitioning is not a viable solution for extreme scientific computing. Instead, we advocate for an alternative approach involving read-only format-aware pre-processing enabling on-the-fly dynamic partitioning.

We present **Dataplug**, an extensible framework that implements the on-the-fly data partitioning model. Dataplug hides the complexities of unstructured scientific data pre-processing and partitioning, offering researchers a data-driven pre-processing and dynamic on-the-fly partitioning strategies for diverse unstructured scientific data formats. The framework provides open-source implementations of pre-processing and partitioning strategies for formats from different domains, inviting developers to contribute or adapt the code to their needs. Dataplug enables efficient parallel access to existing unstructured data blobs in their original scientific format. Currently, we support FASTA, FASTQGZIP, and VCF for genomic data, LIDAR for geospatial data, and imzML for metabolomics data, with plans to include additional formats in the future. Furthermore, Dataplug is compatible with Cloud-optimized formats like Cloud-optimized Point Cloud[14] and Cloud-optimized GeoTIFF[15]. These formats only support content queries, whereas Dataplug brings to them partitioning semantics that simplify their processing in data-parallel workloads. Dataplug's objective is to unlock extreme data processing that was previously unfeasible or prohibitively expensive. Finally, dynamic partitioning enables dataset-wide re-partitioning at zero-cost, allowing one to utilize different partition sizes and to choose the optimal one for each workload.

Architecture. As stated in deliverable *D2.1 Initial Architecture Specifications*, Dataplug is characterized by two elementary phases in its data model:

Cloud-aware pre-processing and indexing. In order to enable parallel access to unstructured scientific data, a pre-processing phase is required. In Dataplug, each supported format is required to define a pre-processing method that must be applied to each raw data blob. This pre-processing method is format-specific, and extracts metadata from the raw data blob, such as internal content structure, indices, and attributes. For semi-structured or tabular formats like JSON or CSV, generic pre-processing methods like schema inference[16] can be applied. However, scientific unstructured data formats require domain-specific tools and techniques to extract valuable metadata. Dataplug aims for an extensible approach, which allows for flexibility in metadata and index structure, as well as in the generation process. We say our pre-processing and indexing approach is Cloud-aware, which means that it is specifically optimized for efficient data access in Cloud object storage. Cloud-aware indexing focuses on exploiting the high-bandwidth capability of object storage so that partitions can be retrieved using many concurrent HTTP GET byte-range requests over large data blobs.

Figure 3 visually represents the indexing difference between Cloud-optimized data formats and our Cloud-aware approach. On one side, Cloud-optimized formats 1) re-arrange the blob's content and 2) embed metadata and indexes within the blob. On the other side, Cloud-aware indexing 1) leaves raw data unmodified, as is, and 2) stores the metadata decoupled from the raw data blob. These two differences are key to why our model is more suitable for Cloud object storage. Since object stores are immutable, transforming files to Cloud-optimized is inefficient, requiring a complete blob rewrite. Instead, our model stores indexes (which are much smaller in volume) in another object, making the pre-processing cost considerably lower.

Data slicing: Dynamic on-the-fly partitioning. After pre-processing and extracting metadata, appli-

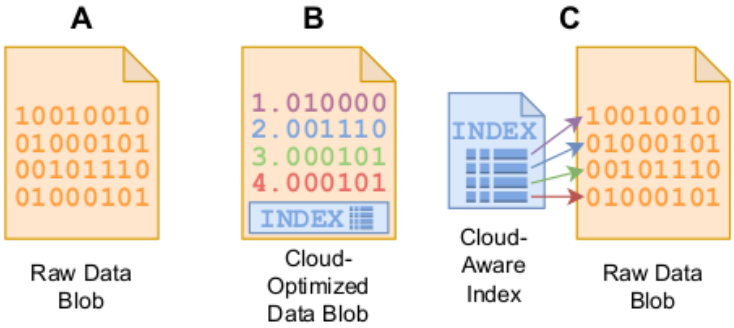

Figure 3: Comparison between Cloud-optimized data formats and Cloud-aware data formats. In Cloud-optimized, metadata is embedded in the blob, while Cloud-aware pre-processing stores metadata decoupled from the blob.

cations may request dynamic on-the-fly partitions of scientific datasets that have been pre-processed using our Cloud-aware approach. We refer to this process as data slicing. A data slice is an entity that represents a lazily-evaluated partition of an unstructured dataset in Cloud object storage. It encapsulates partition metadata (such as byte ranges) and code. When a data slice is evaluated, its embedded code runs and fetches the actual partition data content from Cloud object storage using its native APIs, typically through many concurrent byte-range HTTP GET requests. Additionally, the code can apply any necessary corrections to ensure integrity for the data format, such as adding missing headers. Data slices are generated for a specific dataset by applying a partitioning strategy. In Dataplug, each data format can define multiple partitioning strategies, allowing for different criteria or partitioning parameters. Prior pre-processing of the dataset is necessary so that strategies can query the corresponding metadata and indexes for defining data slices. Data slicing does not involve data movement. Instead, partitioning strategies operate on the metadata and indexes generated during the pre-processing stage, rather than directly on the actual data. Ideally, the index data structure is fully loaded in memory for efficient processing. Once a set of data slices is generated, they may be distributed to remote workers. Each worker independently evaluates a data slice and loads the corresponding data partition into memory. This parallel and distributed process takes advantage of the synchronization-free parallel access and high bandwidth capabilities offered by object stores.

Data life-cycle management. Figure 4 depicts a big picture of Dataplug's architecture:

1. Pre-processing: Raw data (1) is stored as objects in a object storage bucket. This data is in its original, unprocessed raw form. Dataplug requires a pre-processing (2) phase to enable parallel data access to the raw data. When a new object is uploaded, Dataplug can leverage storage triggers to automatically launch pre-processing jobs (for instance, using an Amazon S3 trigger to invoke a Lambda function). During the pre-processing stage, metadata and indexes are generated from the raw data. This extracted information is stored in a separate metadata bucket within the object storage. Each Cloud-aware format can specify the required pre-processing parameters, such as batch or parallel processing, container resources (CPU and memory) and chunk size.

2. Partitioning: After the pre-processing stage, users can leverage Dataplug to partition datasets for specific workloads by applying a partitioning strategy (3) to define data slices. Dataplug utilizes the metadata and indexes generated during the pre-processing stage to query and extract the necessary information for defining these data slices. Once the data slices have been defined, the user can proceed to submit a parallel processing job (4) using a Python distributed computing framework such as Dask[11] or Ray[12]. The user may pass the set of data slices created as input data for the job. The distributed computing framework will handle the deployment of distributed workers and appropri-

[11] `https://www.dask.org/`
[12] `https://www.ray.io/`

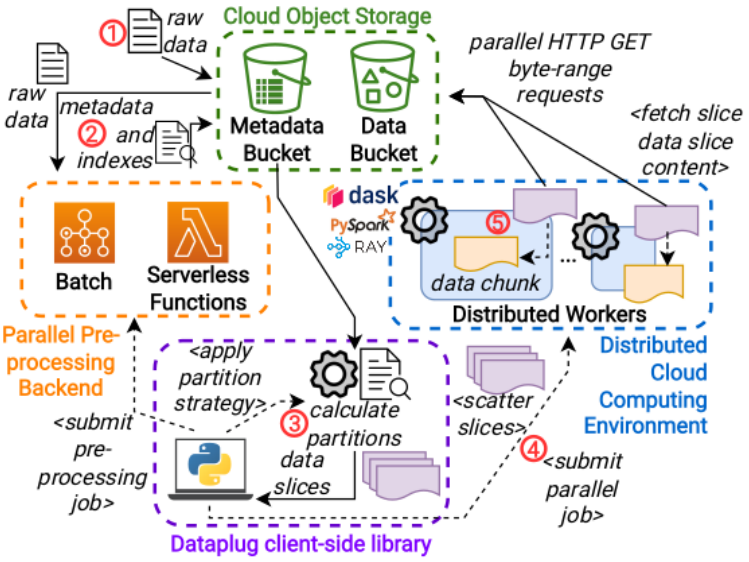

Figure 4: Dataplug architecture.

ately scatter the data slices among them. Finally, each worker process will be assigned a specific data slice as input, allowing them to fetch the content of the data partition (5). Leveraging embedded metadata and code, the data slice can efficiently perform multiple concurrent byte-range HTTP GET requests over one or more objects in the object store. These requests retrieve data chunks.

With the update of the Dataplug component, its API is available in Appendix: 8.1.1 NEARDATA APIs.

4.2.3 HPC Data Connector

In NEARDATA, we are in the process of constructing an HPC data connector to leverage High-Performance Computing platforms into our targeted use-cases. That is, to allow use-cases to use supercomputing facilities, and most particularly MareNostrum supercomputer hosted by BSC-CNS. However the scope of this connector will be limited to a small number of nodes, although tests at larger scale will be attempted. However to do large-scale experimnets we must be awarded specific funds which are managed by independent agencies from Spain and Europe. The task of porting the use-cases into a HPC is a challenging activity given the use-cases of the project are implemented utilizing software typically found in cloud environments (Apache Spark, Lithops, Pravega) and not typically available in a supercomputer. While this grants flexibility, cloud environments computing capabilities are less-performance oriented than those found in supercomputers. Cloud is designed with flexibility in mind and has more relaxed quality of service, while supercomputers are designed in terms of compute power so they require static resources allocations while providing high-performing network, compute and storage capabilities. In this section we describe the work done so far regarding enabling our MDR use-case into MPI and current initial results found. More details of this data connector are described both in *Deliverable 3.1: XtremeHub first relase and documentation* and *Deliverable 5.1: First release of KPI benchmarks in all use cases and data connector libraries*. Additionally, we can find its API in Appendix: 8.1.1 NEARDATA APIs.

4.2.4 Pravega

Pravega is an open-source, streaming storage system designed to handle data reliably and at scale in a distributed environment [17, 18] that is the streaming foundation of NEARDATA. Its architecture is built to cater to the needs of modern stream processing applications, providing seamless ingestion, storage, and retrieval of data streams. At its core, Pravega employs a segmented log model, where

data is organized into independently manageable segments. This design ensures efficient utilization of resources and enables horizontal scalability, allowing Pravega to handle massive volumes of data with ease.

One of the key components of Pravega's architecture is its tiered storage model. This separation enables independent scaling of ingestion and storage resources, allowing the system to adapt dynamically to varying workloads. Pravega employs stream segmentation to partition data logically, enabling parallel processing and efficient resource utilization. Additionally, Pravega incorporates features such as durability, fault tolerance, and exactly-once semantics to ensure data integrity and consistency, even in the face of failures or network partitions. Overall, Pravega's design and architecture make it a robust and scalable solution for building real-time streaming applications in a distributed environment. Note that full details of Pravega's architecture are presented in *D2.1 Initial Architecture Specifications*, whereas a deep performance evaluation and progress beyond state-of-the-art is presented in *D3.1 XtremeHub first release and documentation*.

4.2.5 METASPACE

METASPACE is a cloud software platform for spatial metabolomics. METASPACE incorporates an engine for metabolite identification helping scientists find metabolites in spatial metabolomics data, and hosts a knowledge base of public metabolomics datasets. METASPACE started as a European FP7 project and since then has been funded by both the European Commission and the U.S.A. National Institutes of Health.

METASPACE is an essential software in the emerging field of spatial metabolomics. It is used by almost 2000 scientists from over 100 groups worldwide and hosts the largest collection of spatial metabolomics datasets, with more than 10k annotated samples.

In the EU Horizon2020 project CloudButton, the key computational part of METASPACE, a metabolite annotation pipeline - named `engine` - was implemented to use the Lithops serverless computing framework. This was motivated by the increased use and the need for scalability compared to the previous implementation based on Apache Spark. Input dataset sizes range from a few megabytes to tens of gigabytes, and resource requirements are greatly variable between pipeline stages, so tuning proper resource utilisation with the cluster model became unfeasible. METASPACE switched to using Lithops in production in March 2022.

4.3 Control Plane Components Recap

In this section, we will briefly introduce the components that make up the Control Plane of the NEARDATA architecture.

4.3.1 SCONE

Confidential computing has two essential requirements: 1) CPU auxiliary hardware handling confidential computation and protecting memory; 2) software prepared to use this auxiliary hardware. **SCONE (Secure Container Environment)** makes the bridge between hardware (Trusted Execution Environment – TEE) and software (user application) by making the latter use its libraries that provide the interface to the TEE. SCONE runs the user application in Docker containers, hence protecting CPU and memory in an enclave aided by the TEE. However, network transmission, files handling, operating system interaction, and so on are still subject to interference, despite the fact that the software is ported to SCONE and executed with the help of TEE. To counter that, another mechanism has been introduced.

Confidential computing scope goes beyond the hardware and software individually; it is enforced by the critical decision maker named **CAS (Configuration and Attestation Service)**. CAS will assert that the user application's TCB (Trusted Computing Base)[13] is trustworthy, i.e. its measurement corresponds to the same value configured previously in its policies database. Once the attestation is guaranteed, CAS will deliver secrets (X509 credentials, private keys, etc.), configuration files, file system protection, network protection, environment variables, via a secure channel (protected by

[13]"Set of hardware and software components that are measured to attest the veracity of the execution. More in: https://sconedocs.github.io/glossary/#tcb"

TLS) exclusively accessible by the user application within its enclave. With this setup, the wider scope of confidential computation is guaranteed and enforced.

Albeit wide ranging, the confidential computing scope can be improved. Associate/auxiliary systems can be employed to provide different access levels based on the roles a person or a system can assume. To cover this sort of requirement/functionality, **Keycloak** (an identity and access manager) is used to issue access tokens and corresponding validation tokens to an entity (a person or an application). The system administrator can configure the entity's roles to allow or deny access to different services or levels further on other systems. Keycloak is, among other things, a Single-Sign On system, to which other applications outsource the burden of user management.

4.3.2 AI-based optimizations

In the context of extreme-health use cases, we have massive amounts of varied data. Variety is a key challenge, as it implies the need to use different kinds of resources for proper processing. Moreover, each use case processes and uses data differently. Thus, we find a situation of heterogeneous requirements regarding computing and data collection. Consequently, orchestration policies are needed to distribute that data to the different resources according to the use case requirements.

Traditional approaches apply heuristics to assign resources on demand. However, those heuristics provide static decisions and are not capable to identify changes on resource requests. While this approach allows to meet SLAs in most situations, when there is a peak demand, those SLAs may be impossible to fulfill. Approaches applying heuristics on-demand lack to consider future demand. While the allocation decision might be close to optimal under a given scenario, it may be quite inefficient in a scenario happening a few minutes later after new requests arrive. This situation worsens with the described heterogeneity.

In this component we explore the development of an AI-enabled orchestrator that predicts future demands and requirements and assigns resources based on that knowledge, granting close-to-optimal allocations most of the time. Moreover, the target of such orchestrator is to consider the heterogeneity of resources as well. This heterogeneity is understood in terms of having specialized resources providing performance boosts in some of the use-cases. Some of these specialized resources might not provide a performance boost but be more energy-efficient. That is, they save energy compared to the traditional approach. Consequently, this component will analyze performance of the use-cases not only in terms of computational time but also in terms of energy consumption. Thus, while it might be that a given workload gets the worst time on a given allocation, that allocation might be more energy-efficient.

Summarizing, in NEARDATA the AI-Optimizations will provide a system that recommends the resource allocation for arriving workloads in a smarter fashion that traditional heuristics can do. This component will be integrated in current frameworks (Lithops, K8S orchestrator) to improve the overall QoS of the cloud in terms of speed and/or energy.

4.4 Testbed - KIO Networks

In the following section, we will show the proposed architectures of the Testbeds offered by KIO for three components that make up NEARDATA platform: Pravega (DELL), Lithops (URV) and SCONE (SCONTAIN). From the definition and requirements of the environments provided for the different components, it will be possible to execute the use-cases that take advantage of these technologies.

4.4.1 DELL - Pravega

DELL has requested KIO an OpenShift and an S3 environment to run its testbed within an enterprise environment. The Figure 5 represents the initial testbed architecture for Pravega.

KIO has two distinct environments for deploying PRAVEGA. The first environment, known as the 'OpenShift environment', consists of 3 compute nodes with ESXi, along with its associated vCenter for managing a VMware cluster. This cluster operates on VMware's vSphere 8.0.2. KIO offers two separate solutions for S3 services. One is an S3 service based on NetApp technology. KIO also provides a standalone S3 service based on minIO technology. This service delivers superior perfor-

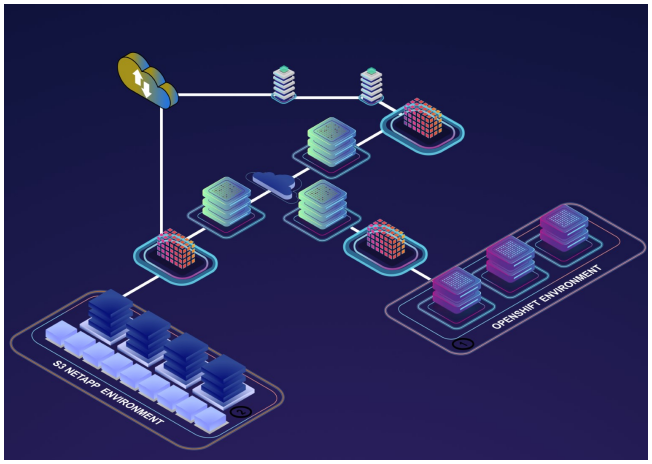

Figure 5: Starting Pravega's Architecture

mance at all levels due to its proximity to the solution. The ability to independently deploy minIO in each environment enhances business model flexibility. This approach allows KIO to offer higher bandwidth and lower latencies to customers, including the PRAVEGA's testbed. For this specific use case, there is no need for a solution that offers high bandwidth and low latencies. This is why we are providing PRAVEGA with the first solution based on NetApp technology, rather than the minIO solution. The hardware requirements for the PRAVEGA's testbed are as follows:

- 108 vCPU 2.3 GHZ.
- 240 GB RAM.
- 1080 GB PREMIUM SSD DISK.
- 50 MEGABIT INTERNET Bandwidth.
- 2 ESP PUBLIC IP: These IP's must hang from the PA-CE.
- 1 VIRTUAL NETWORK : CE-01_OpenShift.
- 16 GB GPU NVIDIA A100 Tensor Core.

OpenShift environment is composed by 3 master and 3 worker nodes with the following technical specifications:

- MASTERS:
  - 12 vCPU 2.3 GHZ.
  - 48 GB RAM.
  - 408 GB PREMIUM SSD DISK.
- WORKERS:
  - 24 vCPU 2.3 GHZ.
  - 96 GB RAM
  - 846 GB PREMIUM SSD DISK
- 1024 GB S3 Object Storage
- Networking Resources:

– 16 GB GPU NVIDIA A100 Tensor Core.

One of the workers has GPU's capabilities with the following technical characteristics: 16 GB GPU NVIDIA A100 Tensor Core. One of the primary requirements requested by PRAVEGA has been to incorporate GPU capability into an OpenShift worker node. This is essential to "off-load" tasks. KIO has two separate environments to address this need, and efforts have been made to enhance the L2 communication level between them. The networking team at KIO had to address this challenge to enable the connection. This solution now enables both testbed environments to communicate with each other, allowing KIO to offer an OpenShift cluster with Nvidia GPU capabilities. The Figure 6 shows the changes introduced in the Pravega's Architecture to add the GPU environment.

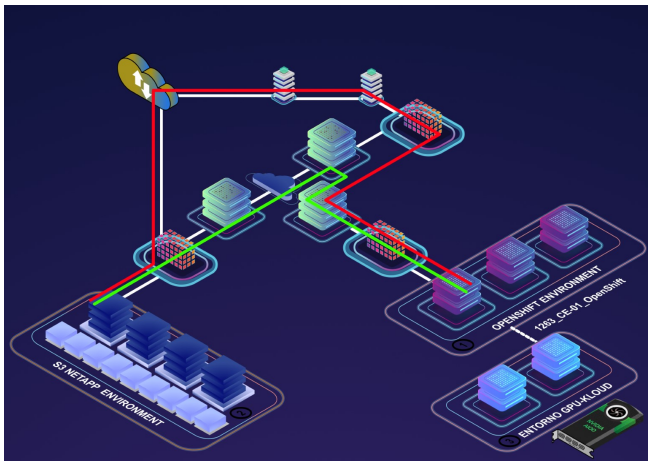

Figure 6: First Evolution Pravega's Architecture

After deploying OpenShift's cluster, we discovered that PRAVEGA's solution was not yet compatible with OpenShift. Despite attempts to adjust permissions, we ultimately decided to switch to a Kubernetes cluster. This Kubernetes environment utilizes identical hardware resources to the previous OpenShift cluster. This transition was made in order to streamline operations, enhance optimization, and reduce latencies, all while eliminating the need to manage two separate environments. It's important to note that the Kubernetes cluster was not deployed within the KK8S environment, but rather directly on the GPU environment. The Figure 7 depicts the final proposed architecture for Pravega with the OpenShift environment modified by the Kubernetes cluster.

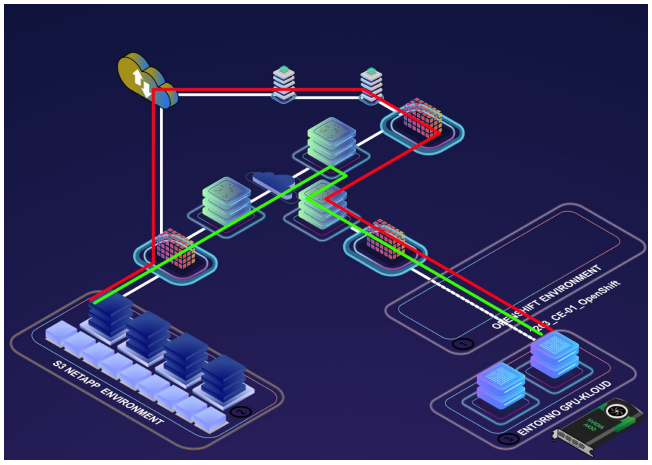

Figure 7: Second Evolution Pravega's Architecture

At the network level, both public endpoints that were previously available are maintained, ex-

tended, and repointed to the new environment. Disk space was increased at the Kubernetes cluster level as described below:

- +200 GB PREMIUM SSD DISK -> Added to Master 1 for "registry"
- +300 GB PREMIUM SSD DISK -> Added to all Workers for new SC

Independent storage drives enable local disk managed by Rancher. Registry is also available to store PRAVEGA's testbed images.

4.4.2 URV - Lithops

URV requested an OpenShift and an S3 environment to run Lithops within an enterprise environment. For this specific use case we first tried S3's Netapp solution, as we did not know how much performance was enough for the testbed that URV was going to run. URV's testbed hardware requirements have been the following ones:

- MASTERS:
  - 12 vCPU 2.3 GHZ.
  - 48 GB RAM
  - 408 GB PREMIUM SSD DISK
- WORKERS:
  - 96 vCPU 2.3 GHZ.
  - 192 GB RAM
  - 672 GB PREMIUM SSD DISK
- 500 GB S3 Object Storage
- Networking resources:
  - 50 MEGABIT INTERNET Bandwidth.
  - 2 ESP PUBLIC IP: These IP's must hang from the PA-CE.
  - 1 VIRTUAL NETWORK : CE-01_OpenShift. Virtual FW PALO ALTO & Management Services

First test on the infrastructure that URV was working on did not meet their expectations. The following explanation is necessary to understand the situation. There are two distinct Internet services in KIO:

- Corporate access: provided by a third-party entity with redundant infrastructure across two different operators, offering a maximum aggregated throughput of 3 Gbps and cloud-based antiDDoS protection by a cybersecurity provider.
- Secondary access: provided by a Telecom Operator with dual circuits in active-passive mode, offering a throughput of up to 2 Gbps, expandable to 10 Gbps. Unlike Corporate access, it lacks an antiDDoS shield.

One of the primary reasons for the delay encountered on URV's testbed was the traffic passing through the Corporate access, leading to increased latency and reduced bandwidth. The Figure 8 depicts the proposed starting URV's Architecture with the S3 NetApp environment.

KIO's Communications team sorted out the issue redirecting the internal traffic to the NetApp environment. This new setup resulted in a fivefold increase in throughput and a decrease in latency.

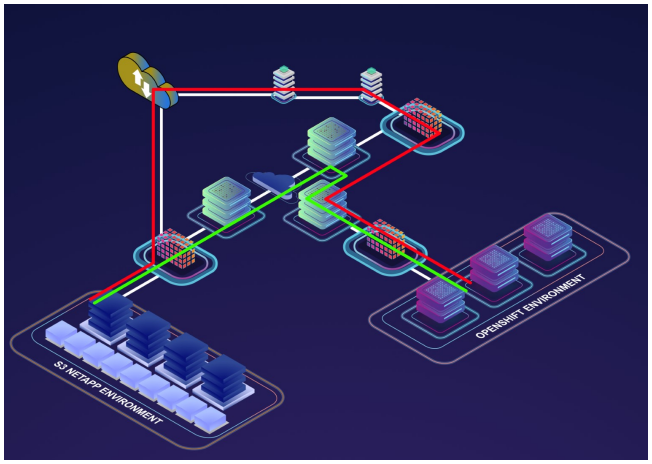

Figure 8: Starting URV's Architecture

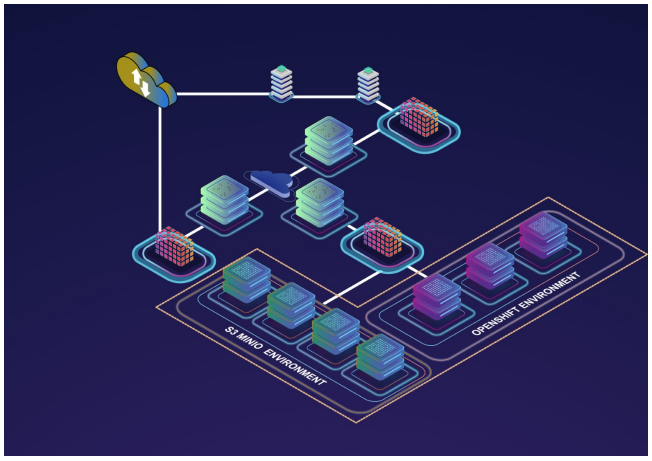

Figure 9: First Evolution URV's Architecture

This was the primary motivation for transitioning from S3's NetApp environment to MinIO's environment. The Figure 9 shows the changes set up to modify the S3 NetApp environment to a MinIO environment in the URV's Architecture.

This MinIO's environment has been deployed with the following resources:

- 16 vCPU 2.3 GHZ.
- 32 GB RAM
- 1672 GB PREMIUM SSD DISK

MinIO's solution has been deployed within the same L2 where the OpenShift environment was deployed. This implementation helps to avoid any penalties in terms of latencies and throughput.

In addition to the infrastructure deployed, URV needed a machine to orchestrate Lithops deployment. KIO has provided for this purpose a Kloud2.0's environment with the following resources:

- 2 vCPU 2.3 GHZ.
- 8 GB RAM
- 450 GB PREMIUM SSD DISK

The Figure 10 shows the final proposed URV's Architecture with the addition of the Kloud2.0 environment for the URV to manage its own resources.

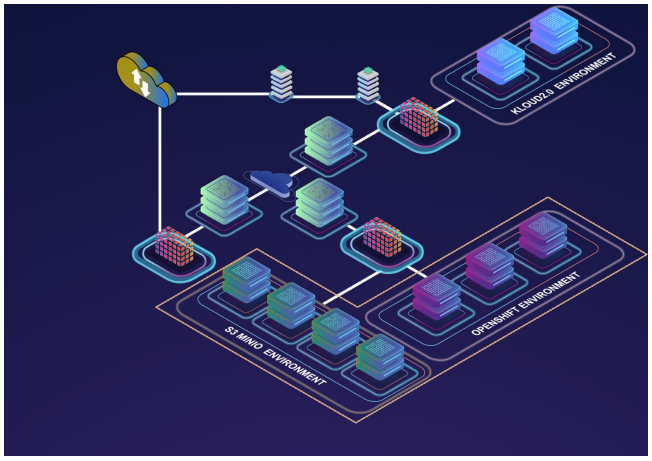

Figure 10: Second Evolution URV's Architecture

4.4.3 SCONTAIN - SCONE

SCONTAIN requested an SGX environment to run their testbed within an enterprise environment. The Figure 11 depicts the starting SCONTAIN's Architecture.

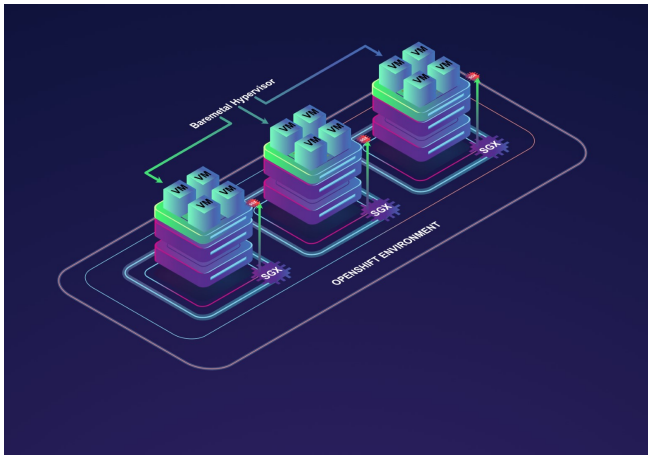

Figure 11: Starting SCONTAIN's Architecture

SCONTAIN's testbed hardware requirements have been the following ones:

- 16 vCPU 2.3 GHZ.
- 64 GB RAM
- 564 GB PREMIUM SSD DISK
- Networking resources:
  - 30 MEGABIT INTERNET Bandwidth.
  - 2 ESP PUBLIC IP
  - Virtual FW PALO ALTO & Management Services

KIO's cloud processors can support SGX. However, this feature has never been enabled for a client in a production environment.

The initial step was to enable it. This process allows for the generation of a key. After obtaining the key, you must register this key within Intel Provisioning Service and receive an EPID Key. Once we have that key, we can share and validate attestation messages. However, a bug was discovered between CISCO and INTEL, making it impossible to properly regenerate the Provisioning Key. After INTEL and CISCO resolved the issue, SCONTAIN was able to effectively utilize the attestation service over KIO's Cloud IaaS.

5 NEARDATA Components Integration

During this section, we will introduce the integrations of NEARDATA components from a theoretical description and a first prototype from the technical specifications of each component. In this way, we present the workflow and data flow between components to understand the interaction of the components of the actual NEARDATA architecture.

It is important to highlight that the documentation, first tutorials and automatic tests of each component and the different integrations can be found in the open Github repositories presented in Appendix: 8.1.1 NEARDATA Github repositories.

5.1 Lithops as a compute engine for Dataplug

As presented above, Dataplug is an extensible framework for partitioning extreme unstructured scientific data. Its two-phase data model allows the user to avoid processing the same data repeatedly and having to duplicate data in the storage backend. Since Dataplug is fully portable, it allows us to use and/or combine different distributed data analysis clusters or serverless frameworks for each of the phases.

The Lithops serverless data processing platform presented at NEARDATA fits perfectly with the features and specifications of the Dataplug data model. Thanks to the compute API offered by Lithops, we will be able to access the compute backend provided by the different cloud providers or NEARDATA's own Testbed offered by partner KIO Networks.

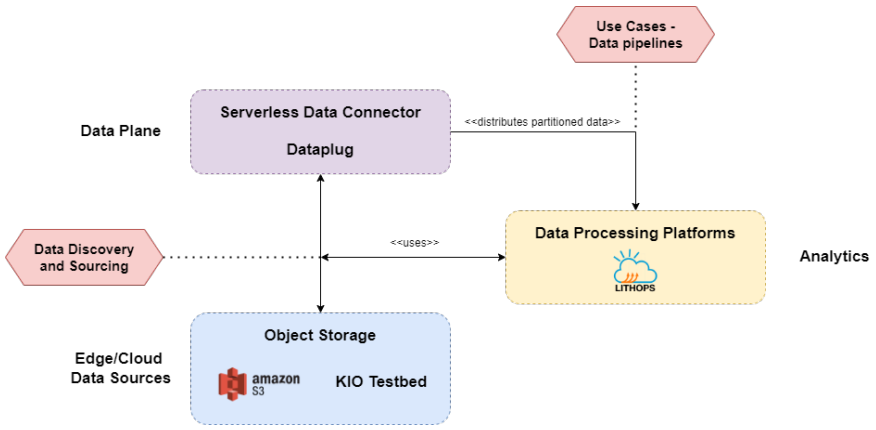

Figure 12: Lithops and Dataplug integration.

Figure 12 shows the theoretical integration of the Lithops and Dataplug components with respect to the architecture proposed in Figure 2 above. The three modules involved in this integration can be identified. The Data Plane, which provides data management on the NEARDATA platform. With our serverless data connector component Dataplug, we provide data indexing and metadata generated from the serverless data processing platform Lithops located in the Analytics block that will access the different Data Sources located in the cloud or in the edge. In this way, discovery and data collection are ensured. Once this step is done, it will be the Dataplug itself that distributes the partitioned data to the computing backend through the Data Programming Abstractions developed by the Data Pipelines.

5.1.1 Early Prototype

Figure 13 depicts the architecture and data lifecycle of the Lithops integration with Dataplug. These do not imply any changes to the architecture presented by Dataplug, as Lithops can be used as a serverless data processing platform for Dataplug. The two phases of the Dataplug data model can be visualized, where Lithops will be in charge of invoking virtual machines or serverless functions in the cloud or in the egde (KIO Networks Testbed) that will perform the pre-processing tasks by directly accessing the data stored in the object storage. In the second phase, the object storage will

be accessed to calculate the data slices that will be distributed to the computing backends through Dataplug. Then, the corresponding fetch will be performed on each of the workers through HTTP GET requests of byte range. Finally, the data portions can be analyzed in the workers.

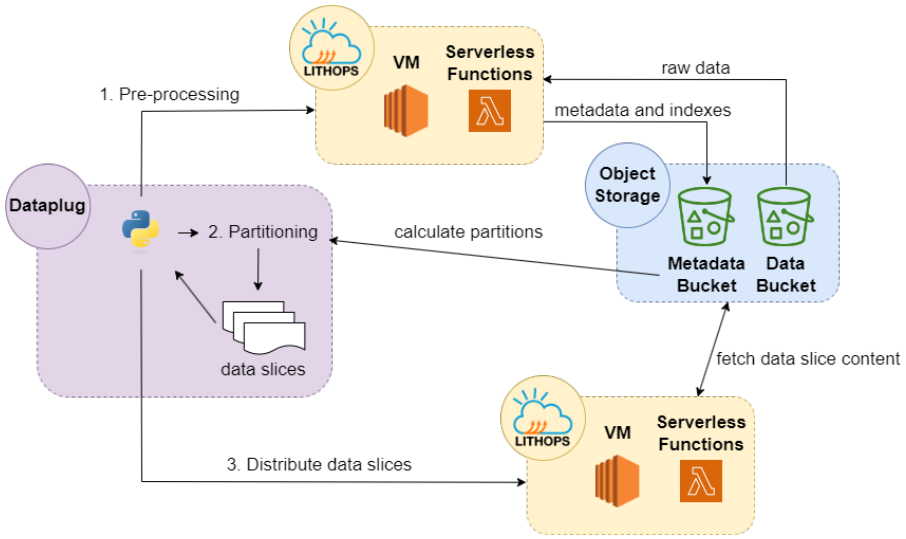

Figure 13: Lithops and Dataplug integration architecture.

Thanks to the portability offered by Dataplug, no modifications are required to use Lithops as a cloud backend. As we have been saying, ease of use is one of the main features of both frameworks, and their combination continues to maintain this attribute.

Figure 14 represents a Python code example to visualize the interaction between the APIs offered by Dataplug and Lithops. The data format to be analyzed is the FASTQGZip compressed genomic format.

```
# Assign FASTQGZip data type for object in s3://testdata/SRR6052133.fastq.gz
co = CloudObject.from_path(FASTQGZip, "s3://testdata/SRR6052133.fastq.gz", storage_config=aws_config)

# Use Lithops as a compute backend for pre-processing
backend = LithopsPreprocessor()
co.preprocess(backend, force=True)

# Generate N batches, using partition_reads_batches strategy
data_slices = co.partition(partition_reads_batches, num_batches=5)

def check_len(data_slice):
    # Evaluate the data_slice
    fastq_batch = data_slice.get()
    return len(fastq_batch)

# Use Lithops for deploying a parallel serverless job
# which will scatter generated data slices, one to each worker
fexec = lithops.FunctionExecutor()
fexec.map(count_lines, data_slices)
result = fexec.get_result()
```

Figure 14: FASTQGZip compressed format example - Lithops integration with Dataplug.

The procedure for parsing this type of data is simple. First, the user must create a `Cloud Object` by entering the format of the file to be partitioned, in this case, FASTQGZip and the key of the file stored in the object store. This cloud object incorporates all the mechanisms to preprocess and partition data transparently.

Lithops will be the backend used to perform the computation necessary for the preprocessing of our data. Within Dataplug there is a specific library fully adapted to use Lithops as a computation

backend. This can be called through the `LithopsPreprocessor()` function. Once that is done, the `preprocess()` function will be applied to the generated cloud object. In this case, the preprocessing strategy is predefined in the Dataplug library, so the user does not have to worry about its management or implementation. This function will generate the metadata and indexes needed to partition the data.

Next, the Data Slices will be generated from the `partition()` function. To do this, the predefined partitioning strategy must be indicated. For the FASTQGZip format, there is a `partition_reads_batches` function, which partitions the data without leaving a half-partitioned read. In addition, the number of Data Slices to be performed must be specified with the parameter `num_batches`.

Once the data slices are obtained, they can be distributed to the different workers used by the computation backend. In this case, Lithops will be used to invoke parallel serverless functions. In order to perform this type of work, Lithops needs to create a `FunctionExecutor` object. From this, parallel serverless tasks can be invoked from the `map()` function. It should be noted that N functions will be invoked according to the total number of Data Slices generated. Each worker will execute the `check_len()` function. This function receives by parameter a generated Data Slice, which is evaluated from the `get()` method. This allows the HTTP GET requests to be made from the metadata stored inside it in order to receive the correct portion. Once the selected batch is available, its size will be returned. Finally, the results of the functions will be collected by the `get_results()` function.

As we have observed, the presented example is a sample code in which Lithops has been used as a computing platform to analyze the partitioned data from the Dataplug Data Connector.

5.2 Lithops & Pravega (Streaming Data Platform)

The Pravega open source streaming storage system is presented in NEARDATA as a streaming data connector. It allows us to process data in streams by providing mechanisms to facilitate the ingestion, storage and retrieval of data streams. Our serverless processing platform Lithops, as we have been discussing in this deliverable, incorporates a Storage API that allows access to data stored in the object storage. Although this provides certain benefits as we have seen in the integration of Dataplug and Lithops, it does not satisfy the need for real-time low-latency data processing. Likewise, the integration of Pravega with Lithops allows us to take advantage of the benefits of Pravega's streaming data management to perform real-time analysis on computational backends deployed with Lithops.

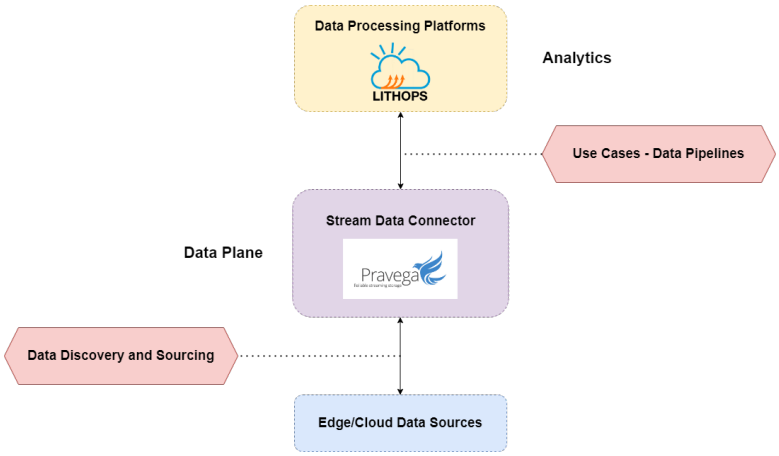

Figure 15: Lithops and Pravega integration.

Figure 15 shows the theoretical integration between Pravega and Lithops. We can see how the workflow incorporates up to three modules seen in the NEARDATA architecture. Lithops belonging to the Analytics module is in charge of processing the data streams ingested in real time from the implemented data pipelines. The Pravega stream data connector belonging to the Data Plane will be in charge of managing the data streams of the data stored in the different Data Sources to ensure data discovery and management.

5.2.1 Early Prototype

In this Proof of Concept, we integrated Lithops in the Dell Streaming Data Platform (SDP) [19], which has Pravega as its core streaming storage engine, to execute multi-cloud serverless functions. The integration of Lithops in SDP has been a natural process, as Lithops has been designed to run functions in containerized execution environments (called "runtime environments"). As Kubernetes is one of the compute back-ends supported by Lithops, we opted for this option to run Lithops jobs in SDP. Fig. 16 shows the main integration points of Lithops with SDP:

- *Object Storage*: Lithops was initially designed to run serverless function jobs on top of object storage. SDP also includes object storage for long-term stream data (*e.g.*, ECS, ObjectScale). One clear integration point of this PoC is allowing Lithops jobs to interact directly with ECS to perform computations.
- *Streaming Storage*: The data stream is SDP's main data storage abstraction. In this PoC, we also work on allowing Lithops to exploit Pravega data streams to performe serverless computations.
- *External Clouds*: One of the key ingredients of Lithops is its multi-cloud capabilities. In this PoC Lithops on SDP accessed and processed data stored in other clouds (Amazon Web Services).

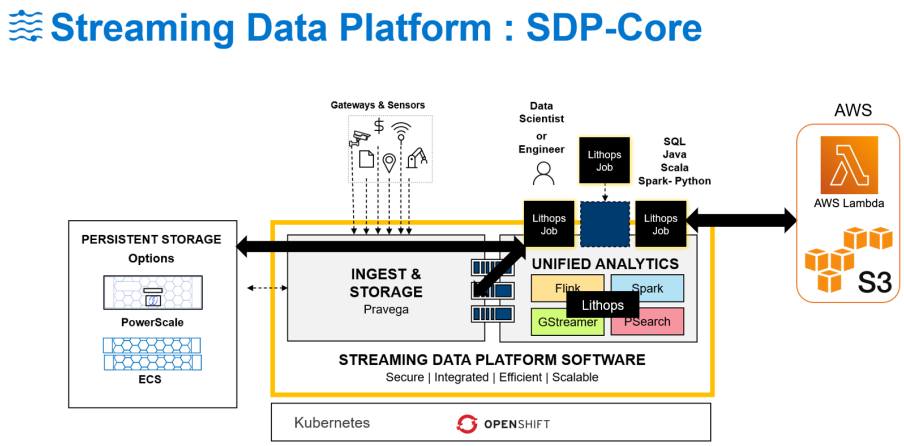

Figure 16: High-level overview of the PoC that integrates Lithops in SDP.

As visible in Figure 16, in our PoC a user can submit Lithops jobs to the SDP cluster. Such jobs are handled in Kubernetes, where each task is executed on an individual pod. Lithops was configured to access the S3 API of the ECS instance within the SDP cluster used in this PoC. This allows Lithops to directly read and write data to the ECS instance. The only addition we had to make in Lithops to allow this was to add the `endpoint-url` option in the configuration of S3. Similarly, providing the right endpoint to the Lithops job enables it to use the Pravega service in SDP. As Lithops is written in Python, we used the Pravega Python client [20] available as a binding on top of the Rust core client implementation [21]. The Python client provide means to manage stream data from Pravega in a Lithops job. Finally, we configured Lithops so it can access an external AWS account. This allows a Lithops job to manage data objects from S3.

Naturally, a few limitations in our PoC should be addressed in case we attempt to make it production-ready it. First, for simplicity, we deactivated authentication and authorization in Pravega so Lithops could interact with it without having to add complexity in terms of security configuration. Second, the Lithops jobs executed in this PoC have been launched from our laptop, which acts as the primary node. In production, it would be advisable to have a primary Lithops node in SDP itself. Finally, the experiments with Lithops have been carried out from the console, meaning that we did not spend efforts trying to integrate a graphical UI to SDP for submitting Lithops jobs. To carry

out this PoC, we had to slightly modify Lithops to add to the S3 configuration a new flag called `endpoint-url` to allow Lithops to work with any on-premises object store that implements the S3 API (we will contribute this change to the project).

Experiment: Multi-cloud serverless `wordcount` job with objects and streams. In this experiment, we extend a `wordcount` job to consume data from Pravega: the streaming storage engine of SDP. That is, the goal of this experiment is to show that we can compute on data from an external object store such as AWS S3 and combine such results with computations on data from Pravega streams. This gives us a sense on the flexibility of Lithops not only in terms of computing across multiple infrastructures, but also using diverse sources of data in the same job. The job used in this experiment is shown in Fig. 17.

```
import random
import lithops
import asyncio
import pravega_client

S3_BUCKET = "s3://lithops-serverless-data/hamlet.txt"
DEFAULT_PRAVEGA_ENDPOINT = "tcp://192.168.200.244:9090"
PRAVEGA_SCOPE = "serverless"
PRAVEGA_STREAM = "wordcount"
PRAVEGA_READER_GROUP = "wordcount_reader_group" + str(random.randint(0, 1000))

def wordcount_object_map(obj):
    counter = {}
    data = obj.data_stream.read()
    for line in data.splitlines():
        count_words_in_line(line, counter)
    return counter

def count_words_in_line(line, results_dict):
    for word in line.decode('utf-8').split():
        if word not in results_dict:
            results_dict[word] = 1
        else:
            results_dict[word] += 1

def wordcount_object_reduce(results):
    final_result = {}
    for count in results:
        final_result = merge_dicts(count, final_result)
    return final_result

def merge_dicts(dict_1, dict_2):
    dict_3 = {**dict_1, **dict_2}
    for key, value in dict_3.items():
        if key in dict_1 and key in dict_2:
            dict_3[key] = value + dict_1[key]
    return dict_3

def wordcount_stream(function_id, pravega_endpoint):
    final_result = dict()
    sm = pravega_client.StreamManager(pravega_endpoint)
    reader_group = sm.create_reader_group(PRAVEGA_READER_GROUP, PRAVEGA_SCOPE, PRAVEGA_STREAM)
    reader = reader_group.create_reader("my_reader")

    data_available = True
    while data_available:
        data_available = False
        slice = asyncio.get_event_loop().run_until_complete(read_events(reader))
        for event in slice:
            count_words_in_line(event.data(), final_result)
            data_available = True

        # after calling release segment, data in this segment slice will not be read again by
        # readers in the same reader group.
        reader.release_segment(slice)

    # Mark the finished reader as offline.
    reader.reader_offline()
    return final_result

async def read_events(reader):
    read_slice = None
    try:
        read_slice = await reader.get_segment_slice_async()
    except:
        print("Problem reading from stream")
    return read_slice


if __name__ == "__main__":
    # Execute wordcount in a bucket with text objects.
    fexec_object = lithops.FunctionExecutor(backend='localhost', storage='aws_s3')
    fexec_object.map_reduce(wordcount_object_map, S3_BUCKET, wordcount_object_reduce)
    # Execute wordcount in Pravega stream.
    fexec_stream = lithops.FunctionExecutor(backend='localhost', storage='localhost')
    fexec_stream.map(wordcount_stream, range(1), extra_args=(DEFAULT_PRAVEGA_ENDPOINT,))
    object_result = fexec_object.get_result()
    stream_result = fexec_stream.get_result()[0]
    # Merge results from both computations
    print(merge_dicts(object_result, stream_result))
```

Figure 17: Wordcount job merging results from AWS S3 and Pravega streams.

The job used in this experiment contains the methods that will be used in the Map/Reduce computations for processing text objects stored in the S3 bucket. Concretely, the `wordcount_object_map()` and the `wordcount_object_reduce()` represent the map and reduce functions to be executed on data imported from AWS S3. On the other hand, we see the function `wordcount_stream()`, which performs similar logic, but assuming that a single function is processing the data stream. It is important to note that Lithops is designed to work on data objects, so the engine abstracts the read/write operations on them. However, using a Pravega stream is new, so we must include the code for reading a Pravega stream in the function's body. We also find the `merge_dicts()` function which combines the dictionaries resulting from the process of counting words in text files, so we output a combined result from both computations.

The `main()` method of the job shows how we trigger the two serverless functions. That is, we run in parallel the Map/Reduce job in a local pod but getting data from AWS S3. Note that, in this example, "localhost" is a pod on the SDP cluster with Lithops installed, not the developer machine. Similarly, we execute the stream `wordcount` function against the Pravega deployment in SDP. In the last lines of the code snippet, we can see that we wait for the futures of both computations and once we get the resulting dictionaries with the word counts, we merge them and print the combined result.

As visible in Fig. 18, when the Lithops job was executed, we could see the read activity from the Pravega metrics perspective. While the amount of data used in this experiment is small (Hamlet text), it is enough to demonstrate our hypothesis. Moreover, Fig. 19 shows the output of the Lithops job from the job driver perspective. At the end of the job execution we can clearly see the combined result of the two `wordcount` functions that used AWS S3 data and Pravega streams, respectively.

Note that this PoC work has been disseminated internally inside Dell, being one key element of our exploitation strategy for Lithops and Pravega. More details on the exploitation ramifications of this effort can be found in deliverable *D6.2 Communication and standardization report*.

Figure 18: Pravega metrics show the reading activity (right dashboards) from a serverless function reading the text of Hamlet written to a Pravega stream.

```
2023-10-19 08:04:23,103 [INFO] config.py:141 -- Lithops v3.0.1.dev0 - Python3.10
2023-10-19 08:04:23,284 [INFO] aws_s3.py:76 -- S3 client created - Region: us-east-2
2023-10-19 08:04:23,632 [INFO] localhost.py:58 -- Localhost compute client created
2023-10-19 08:04:23,632 [INFO] invokers.py:107 -- ExecutorID f9f3f6-0 | JobID M000 - Selected Runtime: python
2023-10-19 08:04:23,682 [INFO] invokers.py:115 -- Runtime python is not yet deployed
2023-10-19 08:04:23,682 [INFO] localhost.py:121 -- Deploying runtime: python
2023-10-19 08:04:25,217 [INFO] invokers.py:171 -- ExecutorID f9f3f6-0 | JobID M000 - Starting function invocation:
wordcount_object_map() - Total: 1 activations
2023-10-19 08:04:25,218 [INFO] invokers.py:207 -- ExecutorID f9f3f6-0 | JobID M000 - View execution logs at /tmp/lithops-
root/logs/f9f3f6-0-M000.log
2023-10-19 08:04:25,221 [INFO] wait.py:98 -- ExecutorID f9f3f6-0 - Waiting for 20% of 1 function activations to complete
100%|
                                                                                                                    | 1/1
2023-10-19 08:04:30,359 [INFO] invokers.py:107 -- ExecutorID f9f3f6-0 | JobID R000 - Selected Runtime: python
2023-10-19 08:04:30,524 [INFO] invokers.py:171 -- ExecutorID f9f3f6-0 | JobID R000 - Starting function invocation:
wordcount_object_reduce() - Total: 1 activations
2023-10-19 08:04:30,531 [INFO] invokers.py:207 -- ExecutorID f9f3f6-0 | JobID R000 - View execution logs at /tmp/lithops-
root/logs/f9f3f6-0-R000.log
2023-10-19 08:04:30,532 [INFO] config.py:141 -- Lithops v3.0.1.dev0 - Python3.10
2023-10-19 08:04:30,539 [INFO] localhost.py:39 -- Localhost storage client created
2023-10-19 08:04:30,539 [INFO] localhost.py:58 -- Localhost compute client created
2023-10-19 08:04:30,539 [INFO] invokers.py:107 -- ExecutorID f9f3f6-1 | JobID M000 - Selected Runtime: python
2023-10-19 08:04:30,556 [INFO] invokers.py:171 -- ExecutorID f9f3f6-1 | JobID M000 - Starting function invocation:
wordcount_stream() - Total: 1 activations
2023-10-19 08:04:30,657 [INFO] invokers.py:207 -- ExecutorID f9f3f6-1 | JobID M000 - View execution logs at /tmp/lithops-
root/logs/f9f3f6-1-M000.log
2023-10-19 08:04:30,660 [INFO] wait.py:98 -- ExecutorID f9f3f6-0 - Getting results from 2 function activations
100%|
                                                                                                                    | 2/2
2023-10-19 08:04:33,804 [INFO] executors.py:593 -- ExecutorID f9f3f6-0 - Cleaning temporary data
2023-10-19 08:04:33,820 [INFO] wait.py:98 -- ExecutorID f9f3f6-1 - Getting results from 1 function activations
100%|
                                                                                                                    | 1/1

2023-10-19 08:05:02,875 [INFO] executors.py:593 -- ExecutorID f9f3f6-1 - Cleaning temporary data
{'THE': 3, 'TRAGEDY': 2, 'OF': 4, 'HAMLET,': 2, 'PRINCE': 2, 'DENMARK': 2, 'by': 92, 'William': 1, 'Shakespeare': 1, 'Dramatis': 1,
'Personae': 1, 'Claudius,': 2, 'King': 34, 'of': 625, 'Denmark.': 7, 'Marcellus,': 4, 'Officer.': 1, 'Hamlet,': 27, 'son': 11, 'to':
608, 'the': 929, 'former,': 1, 'and': 680, 'nephew': 3, 'present': 4, 'king.': 3, 'Polonius,': 6, 'Lord': 13, 'Chamberlain.': 1,
'Horatio,': 15, 'friend': 4, 'Hamlet.': 25, 'Laertes,': 16, 'Polonius.': 14, 'Voltemand,': 4, 'courtier.': 7, 'Cornelius,': 2,
'Rosencrantz,': 6, 'Guildenstern,': 5, 'Osric,': 4, 'A': 87, 'Gentleman,': 1, 'Priest.': 3, 'officer.': 2, 'Bernardo,': 2,
'Francisco,': 2, 'a': 453, 'soldier': 2, 'Reynaldo,': 2, 'servant': 2, 'Players.': 5, 'Two': 4, 'Clowns,': 2, 'gravediggers.': 1,
'Fortinbras,': 6, 'Prince': 1, 'Norway.': 1, 'Norwegian': 1, 'Captain.': 1, 'English': 2, 'Ambassadors.': 2, 'Getrude,': 1, 'Queen':
```

Figure 19: Output of the Lithops jobs that combines the wordcount results from S3 objects and Pravega streams.

5.3 Lithops as a compute engine for METASPACE

METASPACE[22] provides an end-to-end annotation pipeline for Mass Spectrometry Imaging (MSI) datasets. Datasets can be uploaded directly by the user to the METASPACE portal[14] and annotated using only cloud resources. Its lithops-based serverless architecture allows annotation of highly variable input sizes, from a few megabytes to tens of gigabytes. Annotation results are published to the METASPACE portal and made available to researchers around the world.

Often, omics researchers are interested in processing personal and/or confidential data. To meet the privacy requirements of personalised annotation jobs, our goal is to run the METASPACE annotation code in secure enclaves instead of stateless serverless components. Securisation will come from SCONE containers[23], scheduled with Lithops on a K8S backend.

We will rebuild the METASPACE annotation pipeline according to the NEARDATA architectural guidelines. Figure 20 shows how the different components will scale and interact. Users will upload datasets to the METASPACE portal, which will act as the prototype's exclusive data source. Lithops, the heart of NEARDATA's analytics module, will schedule and launch pipeline tasks in a K8S orchestrator running on the KIO testbed. Instead of standard Docker containers, the Data Plane will be

[14] https://metaspace2020.eu/

implemented with SCONE containers, secured with Intel SGX.

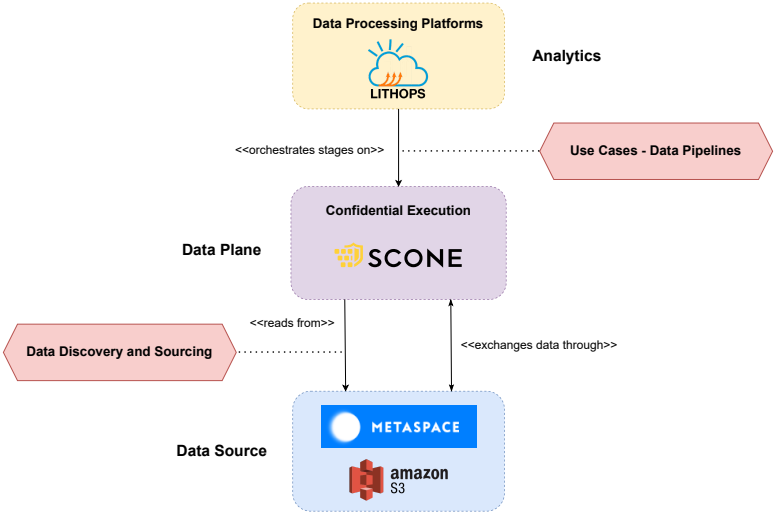

Figure 20: Lithops and METASPACE integration.

5.3.1 Early prototype

Spatial metabolomics consists of applying spectrometry measurements to a biological tissue section. The resulting dataset contains a series of arrays - the spectrometry results - each corresponding to a pixel of the section. Defining the presence and relative abundance of different molecules in each pixel of the dataset is called an annotation. The METASPACE annotation pipeline does this by comparing these results with databases of real metabolite formulae with known spectrometry signals.

In Figure 21 we can observe our first prototype running the METASPACE annotation process on the K8S backend on the KIO testbed. The whole implementation is wrapped into a `Pipeline` class, which comprises the Lithops executor, a log extractor for execution profiling and a cache system to avoid recalculating already processed data, such as a same database on different dataset annotations. All stages of the pipeline are implemented with one to a few consecutive lithops `map` calls, each function running in a exclusive SCONE container.

The pipeline receives two inputs: the dataset to be annotated - in imzML format - and a database to compare against. The dataset is manually uploaded to the METASPACE portal by the user and automatically stored in an S3 bucket. Annotation functions then read it from S3. The database is generated by specifying the set of possible molecules - and derivatives - and calculating their theoretical spectrometry signal with the `pyMSpec`[15] library.

Once the data has been loaded, the exchange between the stages takes place via a private MinIO storage system in the KIO testbed. Intermediate results are not exposed to the cloud throughout the pipeline for privacy reasons. I/O operations with MinIO can be conveniently performed using Lithops' `Storage` abstraction.

The pipeline starts with some preprocessing steps to sort the dataset and the database. The metabolite formulae in the database are sorted (`build_database()`) before their respective spectrometry signals are generated (`calculate_centroids()`). As their overall size is less variable across runs and smaller than that of the datasets, sorting is performed with a fixed number of 32 functions and segmented into 256 final partitions. The dataset is sorted with adaptive parallelism instead, with the number of functions increasing linearly with dataset size and the final partition kept at 128MB (`split_ds()` and `segment_ds()`). The preprocessing ends with a database resort and repartitioning based on dataset segments (`segment_centroids()`).

The annotation *per se*, as the comparison of dataset and database signals, is performed afterwards (`annotate()`), with an embarrassingly parallel computation partitioned based on dataset segments.

[15] https://github.com/alexandrovteam/pyMSpec

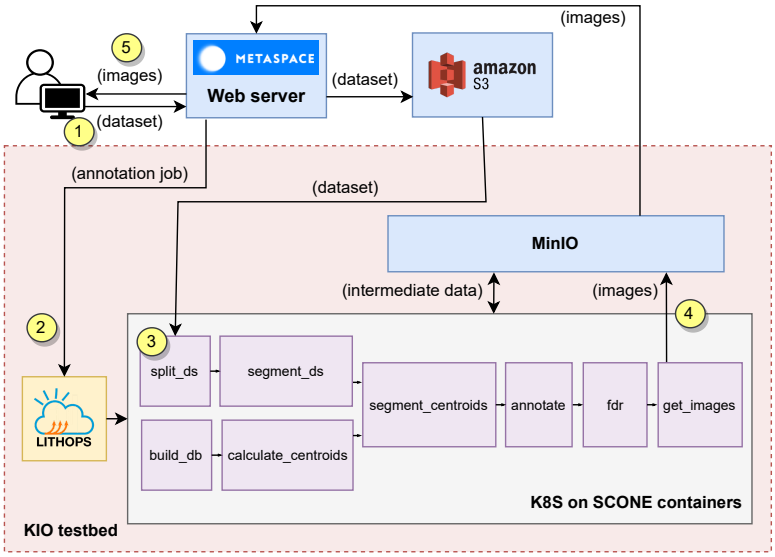

Figure 21: Integration architecture of Lithops and METASPACE. 1) The user uploads a dataset to the METASPACE portal. 2) The job is scheduled in a Lithops client in the KIO network testbed. 3) Pipeline stages are executed in K8s on SCONE containers. Functions read the input dataset from S3 and exchange the intermediate data with MinIO. 4) Resulting images are stored in MinIO. 5) The user accesses the images via the METASPACE portal.

`annotate()` is the most parallel and computationally intensive stage. Finally, the annotated data is statistically validated using a parallel False Discovery Rate algorithm. The signal quality of plausible metabolites is compared with signals of alternative formulae that cannot be present in biological tissues (`run_fdr()`). Only metabolites with the highest statistical confidence are retained in the final results.

The results are formatted into a `png` image per metabolite, showing its abundance at each pixel of the tissue. Images are stored in MinIO and can be viewed and/or downloaded from the METASPACE portal.

5.4 AI-based optimizations for Lithops

The AI-based optimization component allows to orchestrate current frameworks such as Lithops or K8S to improve efficiency and quality of service in terms of speed and/or energy. The Lithops framework offers different APIs to access or deploy services and resources from different cloud providers. During this process, certain characteristics and performance metrics are stored for later analysis. Likewise, we found a potential integration between the two components to leverage the performance metrics generated in Lithops to improve resource allocation for demanding workloads.

Figure 22 shows the architecture of the theoretical integration between the two components. In this case, we see how the Control Plane and Analytics modules are integrated in a simple way.

5.4.1 Early Prototype

The first steps undergone towards this goal have been based on K8S as infrastructure management. The choice was made based on the fact this resource manager is widely spread and production-ready, enabling us with a stable environment from which we could get relevant system telemetry (CPU, memory, disk, etc.) in a reproducible manner.

One of the first steps was done collecting the metrics from the Kubernetes itself. However we realized Kubernetes time window to gather data was too wide and aggregated within the time range. This resulted in extremely inaccurate data and produced predictors with near 100% accuracy. However such accuracy was unrealistic. Consequently we did experiments with Linux monitoring tools in each of the nodes. Despite this technique not allowing us to separate utilization from containers,

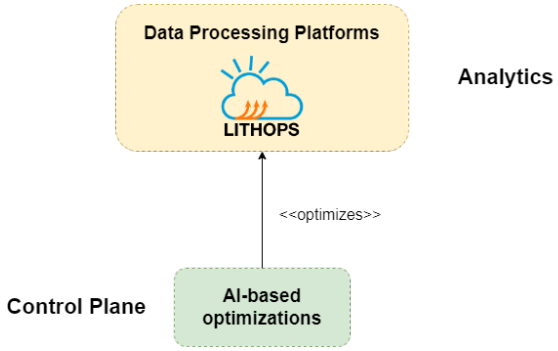

Figure 22: AI-based optimizations and Lithops integration.

we can actually know which containers are running where and we can ensure that each node only runs one processing container at once, so our data is relevant. Given that Linux monitoring tools allow for a much smaller and fine-grained observation window, that data was much more useful.

Figure 23 and 24 shows current work utilizing the Linux monitoring tools with a SVR model as well as a linear regression. We can appreciate how our model predicts utilization for up to 3 future observation windows. That is, from one to three observation windows ahead of time. At the top we present the training sets and how well we do estimate to predict the future, while at the bottom is our test case used to estimate the accuracy of the model.

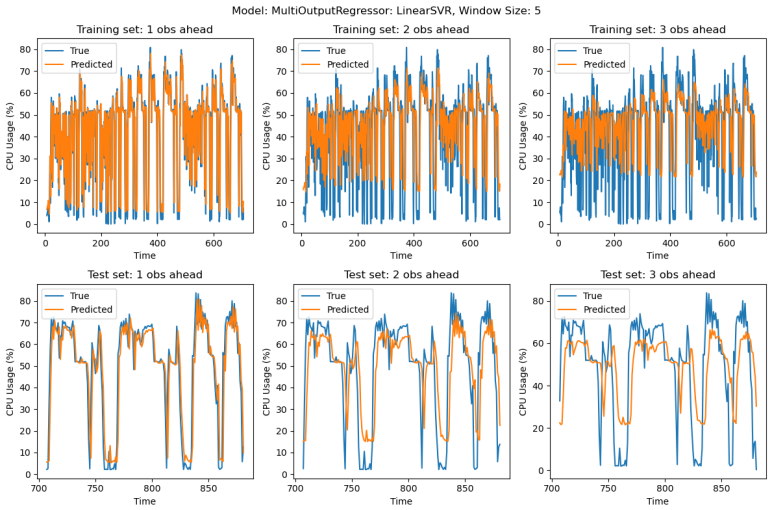

Figure 23: Prediction of future resource utilization using the SVR model

On the other hand, Figure 25 shows the results obtained using Kubernetes telemetry, where we can appreciate how the granularity of the metrics is too wide resulted in unrealistic data.

Those charts represent models being fitted for each of the future observation windows. However, each one does not utilize the information obtained from the previous one. That is, we have a model trained to predict one future window, and a different one to predict two in the future and so on. While this approach allows for proper exploration of the work, we do target to use the previous model to train the predicting one more steps ahead of time. On the other hand, we intend not to fit the model to a specific case but rather build a general one.

Next step is to enable a recommender that gathers all the telemetry information of an application and suggests the best resource allocation based on the currently available infrastructure. To achieve that, the telemetry data is treated as a time series from which we can predict resource utilization in the immediate future. Such predictions can be later used in order to dynamically scale the amount

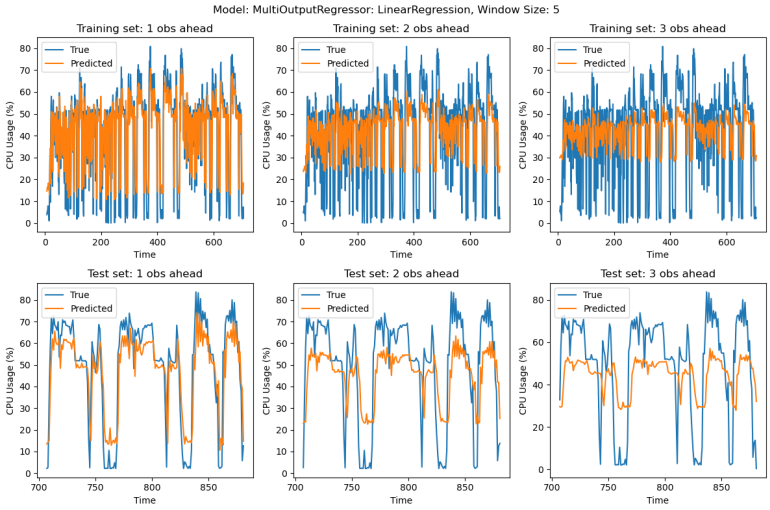

Figure 24: Prediction of future resource utilization using a linear regression

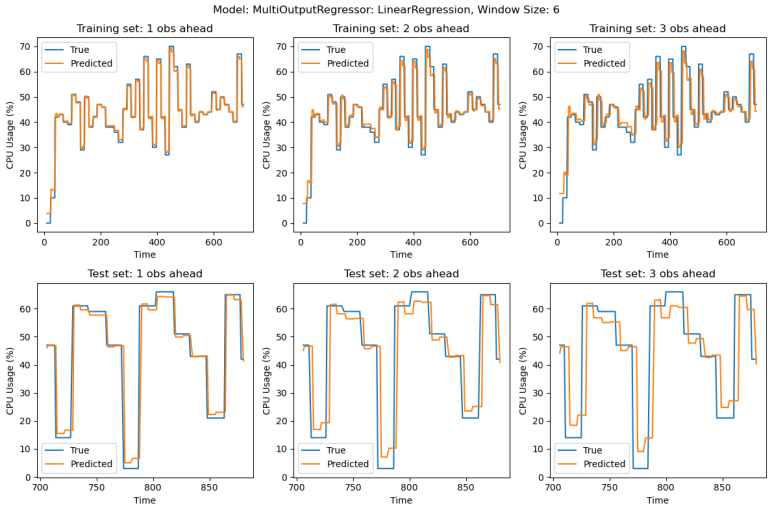

Figure 25: Prediction of future resource utilization using a linear regression with kubernetes-provided telemetry

of resources assigned to the application, freeing unused resources or asking for more of them in anticipation of spikes or drops in the resource usage pattern.

There are situations, however, in which the resource usage is near constant, making such time series recommendations not useful. To solve such situations the system can be expanded with a component identifying the shape of the time series.

Currently we are using Apache Spark on top of Kubernetes to develop the recommender. The choice is made as with Spark we can represent a reasonable amount of real-world use-cases and we can easily reproduce the results. However, as earlier stated, in the future this recommender will be integrated within the Lithops framework, so that we can apply it to all the use-cases developed in NEARDATA.

The connectors we initially identified to implement and integrate such recommender are described as follows:

- **Machine-learning training connector**: this connector will learn from the time series obtained from running workloads, and output the trained model.

- **Machine-learning recommender**: this connector will be directly used by the orchestrator, which will decide which resources to use based on these recommendations.
- **Resource management connector**: this connector will provide several resource management policies. As an input, it will receive workload requirements in terms of resources, as well as SLA requirements to be met. As an output, it will deliver the proposed resource allocation.

The future integration of those procedures with the Lithops framework is depicted in Figure 26.

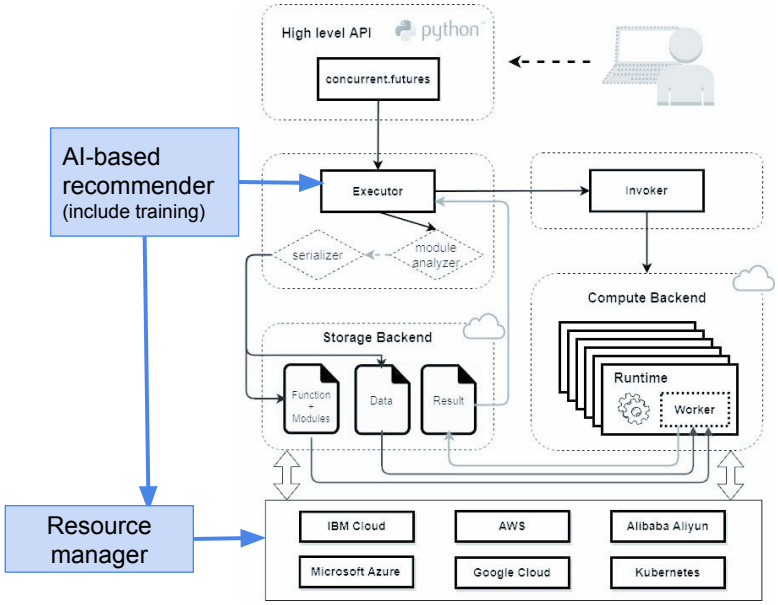

Figure 26: AI-enabled recommender for Lithops

We have already performed initial explorations running benchmarks such as matrix multiplication and the approximation of pi as well as the flops benchmark on top of the Lithops framework. As a result of those initial experiments, we were able to properly identify the target telemetry information we want to initially use as well as identify the main challenges. One of such challenges is the fact that some workloads have a near-constant resource utilization, thus making such a recommender system useless: one can predict how much resources will be used in the future from straightforward observation. Subsequently, we explored more complex workloads leveraging Spark on top of the Kubernetes system. The potential areas of optimization within the Kubernetes framework have been found into the AutoScaler that Kubernetes upon. This auto-scaler is depicted in figure 27.

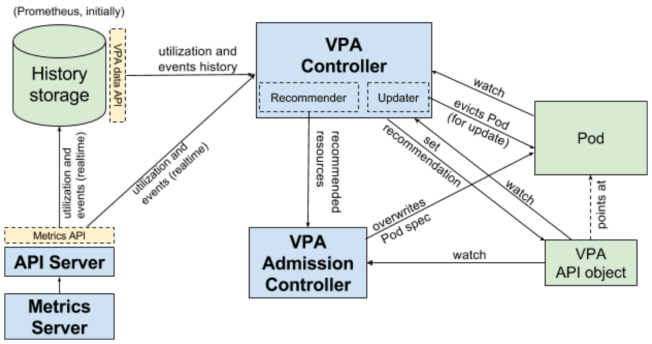

Figure 27: Kubernetes VPA Auto-Scaler on which we aim to integrate our recommender system

This element makes static predictions to decide the resources in often naive and ineffiicent ways. Thus, we aim to integrate our recommender in such system to demonstrate a better decision-making

on top of the Kubernetes platform for Spark workloads. Potentially, that might be applied to genomics workloads too.

When it comes to modelling the recommender itself, two main approaches are being proposed:

- **Real-time streaming**: while telemetry of a running workload is being collected, process the data in the form of a moving window. Within this window a model is filled utilizing a linear regression of a linear SVM to make a prediction of the future time steps based on the observed previous time steps. This approach is undergoing testing and we expect to provide initial results by the next period.
- **Collecting telemetry from several workloads**: we will explore whether neural networks models on large dataset using a variety of workloads can yield better predictions than simpler real-time models. However, since neural networks are costly due to their size and computational requirements, we'll incorporate an additional intermediate component. This component will use clustering of the real-time series (for instance, using encoders) to see whether the workload has a constant or heterogeneous resource utilization pattern. In the latter case, we would switch to using the neural-network based model.

5.5 XtremeHub - Main components

The XtremeHub features the integration of the components that form the Data Plane module. They incorporate high performance data connectors that are perfectly suited to deal with extreme data combining batch and low-latency stream processing deployed in secure and confidential environments such as TEEs.

The XtremeHub presented at NEARDATA consists of the following main components: SCONE as XtremeHub Security to provide secure environments for the confidential execution of data processing platforms and data connectors, Pravega as XtremeHub Streams to guarantee real-time, low-latency streaming data processing and finally, Lithops as XtremeHub Compute. Although this component is not included in the Data Plane, its storage and compute APIs allows to provide serverless computational resources by leveraging a continuous data stream through direct connection to the object storage.

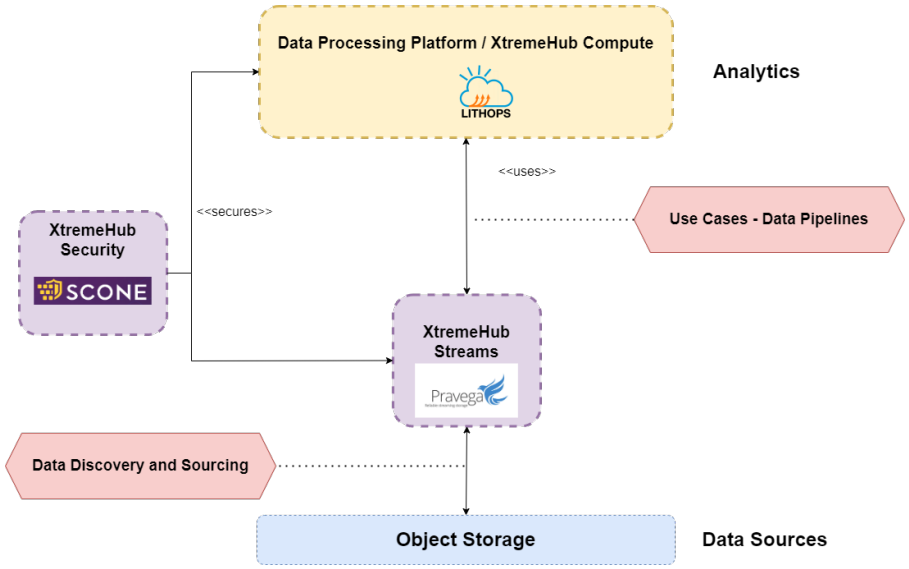

Figure 28: XtremeHub Main components integration (Lithops, Pravega & Scone).

Figure 28 depicts the theoretical integration of the three main components of the XtremeHub. In addition, the use cases leverage the features provided by the XtremeHub to develop and execute sensitive extreme data pipelines from the containerization of processing platforms and data connectors

in trusted environments. In the deliverable *D3.1 XtremeHub first release and documentation* the new NEARDATA XtremeHub is presented in more detail and in depth.

5.5.1 Early Prototype

Applications will reside on SCONE prepared images, (e.g. Python for Lithops). In other words, SCONE is the underlying platform that provides the interface between the ported application and the TEE. With this in mind, the ported applications will run on a confidential Docker container, i.e. CPU and memory access are protected by the TEE.

The enforcement of confidential computing is made via Configuration and Attestation Service, which, via attestation, will provide configurations related to XtremeHub interactions, storage encryption, and other configurations that will only be available to authorized applications or users.

A confidential computing deployment is called "*mesh of services*" and all communications and storage are encrypted or authenticated with keys only provisioned through attestation.

Attested Applications Policies. Policies are the essential data structures used to enforce confidential computing. In brief, they have the following sections (among others): `services`, with the set of services description (name, command and parameters, environment variables, enclave hash); `images`, with description of protected volumes used by the services; and `secrets`, containing certificates, private keys, passwords etc.

These policies are configured prior to system operation. Some secrets, like passwords, can be automatically generated and no *root-level user* will have access to it. The `secrets` can also contain configuration files, e.g. `/etc/my.cnf` for MariaDB; binary-type information, e.g. `*.JKS` files; certificates issued by third parties and so on.

XtremeHub and the other systems within the scope of confidential computing will get their configuration from policies after successful attestation against CAS. Among the configurations mentioned above, credentials to transfer data via secure channel with mutual-TLS (mTLS) is one method to enforce privacy, where only authorized endpoints can receive or send data.

Implementation. A mesh of services has been deployed in a Kubernetes cluster and consists of: CAS, Lithops (operating in "*localhost*" mode), Certifier, and Keycloak. A user application is also used on the desktop-side to demonstrate how an authorized person (with username and password) can login to Keycloak, issue an access token that will be validated by the Lithops program and have the subsequent execution triggered.

The Certifier is a gateway to CAS. In this POC scenario, the Certifier will provide the respective certification authority (CA) certificate from which the Keycloak's server certificate was issued – assuming the CA certificate is not available in the public key infrastructure (PKI[16]). The Certifier is a micro-service application that can be accessed from the browser and the corresponding CA certificate will be downloaded in PEM format. This CA certificate can be imported into the browser or the operating system certificates manager. However, if all the systems and endpoints are covered by the confidential computing enforcement measures, the Certifier is not necessary. Furthermore, if a CA certificate has been issued previously by a valid registrar (or registration authority – RA), it can be injected into the policies and used to issue server certificates within CAS in which cas browsers will get CA certificates from PKI instead.

Lithops in this POC is not running complex processing or long jobs, but has a demonstration of parallel computation running on the Docker container based on a SCONE image. Figure 29 shows the workflow, since attestation, progressing to phase where users authenticate themselves in Keycloak, issue access tokens, forward them to Certifier in order to get secrets, and finally submit processing to Lithops. Systems (Certifier and Lithops) will grant or deny execution by validating the access token, and proceed accordingly. For this POC, the role "**nd-role**" is used to grant authorization.

Another POC was tested where Lithops on client-side is configured in "*serverless*" mode and the program submits jobs to a Kubernetes cluster with TEE support to run Lithops functions in confi-

[16]PKI: https://en.wikipedia.org/wiki/Public_key_infrastructure

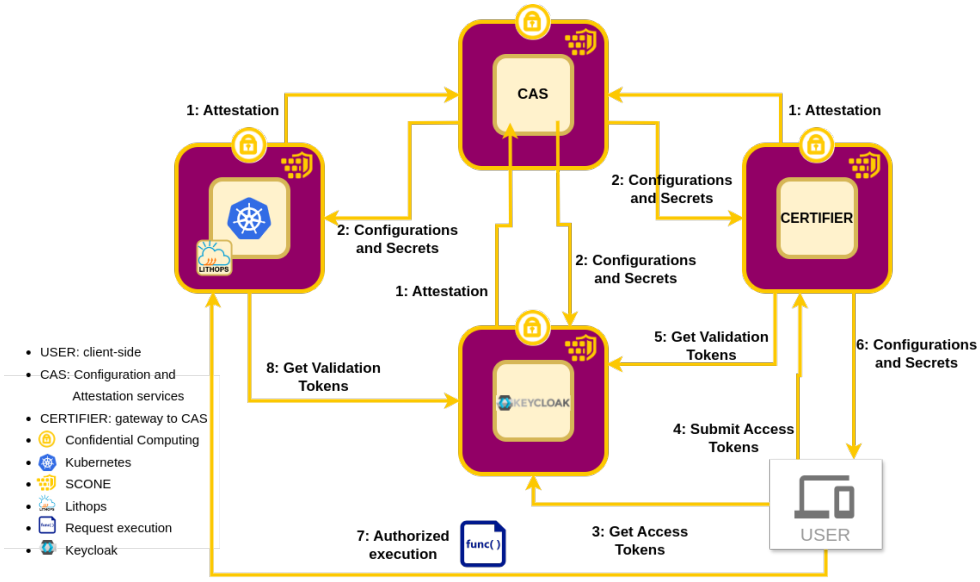

Figure 29: Confidential computing mesh of services.

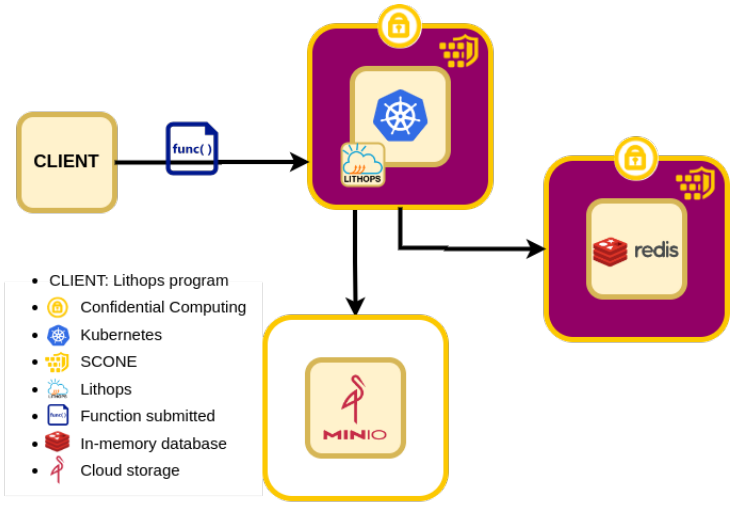

Figure 30: SCONE Lithops execution in serverless mode.

dential computing environment. It is composed by a Kubernetes cluster providing access to SGX hardware device; a Redis server, the worldwide used in-memory database, and also a minIO, an equivalent to S3 cloud storage system. The Figure 30 depicts this POC. Some modifications had to be made in Lithops source code in order to enable a succesfull deployment of new containers via jobs dispatching. For example, the source code *lithops/serverless/backends/k8s/config.py* now has its variable **JOB_DEFAULT** modified with the following SCONE configurations:

```
...
    env:
      - name:  ACTION
      value:  ''
      - name:  DATA
      value:  ''
      ...
      - name: SCONE_HEAP
```

```
      value: '768M'
      - name: SCONE_MODE
      value: 'AUTO'
      - name: SCONE_FORK
      value: '1'
      - name: SCONE_CAS_ADDR
      value: '172.20.0.1'
      - name: SCONE_LAS_ADDR
      value: '172.20.0.1'
...
limits:
   cpu: '8'
   memory: 8192Mi
   sgx.k8s.io/sgx: "1"
```

Environment variables "SCONE_LAS_ADDR" and "SCONE_CAS_ADDR" are host addresses where the SCONE attestation mechanism validates the enclave and retrieve the corrsponding policies. The parameter "limits: sgx.k8s.io/sgx: "1"" tells Kubernetes to provide SGX device to that container; and, along with "SCONE_MODE=AUTO" or "SCONE_MODE=YES" the ported system will benefit from the TEE. Variable "SCONE_FORK=1" is essential for Lithops to work with multiple concurrent processes. And "SCONE_HEAP" is the minimum size reserved for the application startup. There is one variable not configured now: "SCONE_CONFIG_ID". It contains the *policy name + service* to validate the enclave and get the corresponding secret configurations. Presently, attestation in serverless mode is under development. It can be seen above that the values on the SCONE_* environment variables are all *hard-coded*. The analysis to implement attestation here points to include a new section in the Lithops execution configuration file (normally located in /etc/lithops/config), the "scone:" key. For example:

```
lithops:
   ...
k8s:
   ...
scone:
   scone_mode: AUTO
   scone_config_id: xtreme-lithops-43245-19512-79041/mapgenomics
   scone_cas_addr: 10.185.25.37
   scone_las_addr: 172.20.0.2
   ...
```

The limits: configuration section had to be increased. SCONE requires a "larger room" for operating enclave's cryptography functions and memory management. In server applications, either the limits: memory: is omitted or set to the cluster's threshold. We expect to have these and other variables be dynamically set in Lithops configuration file instead of having to recompile a Docker image with specificities. Additionally, Lithops prepares YAML manifests for Jobs and Containers dynamically and relies on dynamically set environment variables, such as MASTER_POD_IP to be present in container at startup. No variable, with the exception of a few SCONE_* are read prior to attestation. All the environment variables that are visible to the program come from the policies after attestation. And the policies must be uploaded before the execution, via another workflow.

It is important to make clear that confidentiality is maintained in Lithops container, regardless of the backend being localhost or serverless (please refer to D3.1 XtremeHub first release and documentation, section 6.2 Lithops Experimental Execution). The next steps will focus on enabling attestation for Lithops in serverless mode; fine-tune software and configurations, and address bugs and inconsistencies as soon as they appear.

6 NEARDATA Health Data Spaces

From the NEARDATA platform we present three International Health DataSpaces to promote their worldwide adoption in the fields of Metabolomics (thanks to EMBL), Genomics (starting with BSC, SANO and UKHS) and Surgery (thanks to the Dresden network).

During this deliverable we have been exposing the technologies developed, its integration and its function within the NEARDATA platform. Next, we will introduce its adaptation to deploy and validate the three International Health Data Spaces.

6.1 Metabolomics Data Space

The NEARDATA Metabolomics Data Space aims to become a worldwide reference in the field of metabolomics. We take into account the considerable efforts that make EMBL a leader in this field to leverage the METASPACE platform as the main component for spatial metabolomics data analysis, sharing, and visualization.

Its widespread adoption by hundreds of scientists around the world makes METASPACE a major point of interest. By adopting the NEARDATA platform, scientists will be able to benefit from data analytics features such as data discovery, management and processing with the ability to secure confidential workflows.

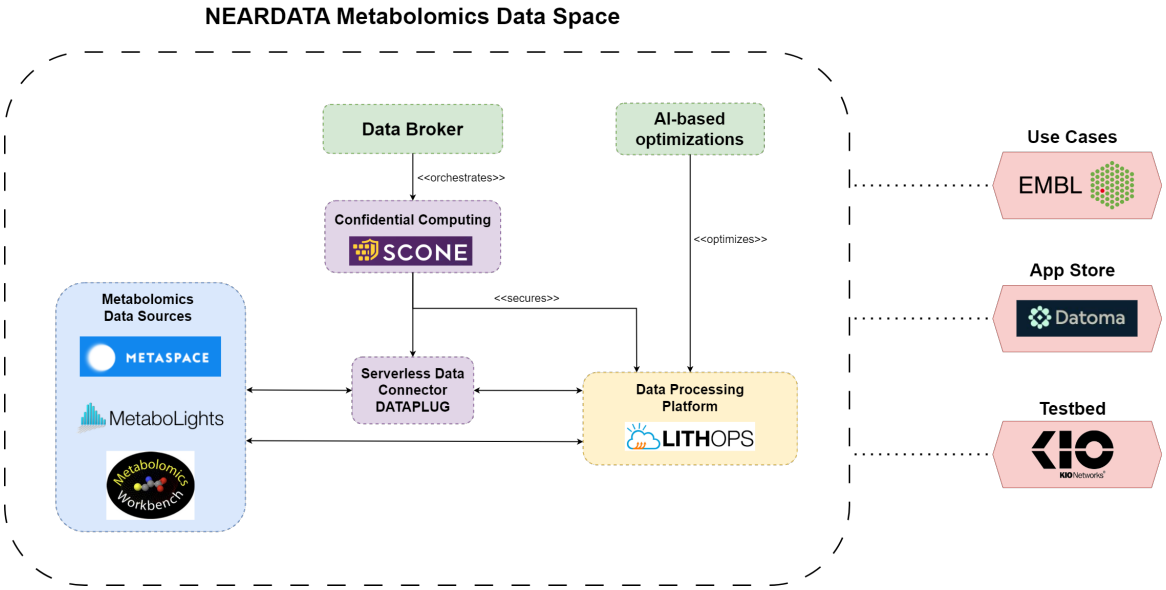

Figure 31: NEARDATA Metabolomics Data Space.

Figure 31 shows the NEARDATA Metabolomic Data Space. We can identify the Data Broker and AI-based optimizations components belonging to the Control Plane module, the SCONE and DataPlug components available in the Data Plane module, the Lithops component of the Analytics module and finally, the core of the NEARDATA Metabolomic Data Space which is METASPACE incorporated in the Data Sources module. Also, the EMBL use case on spatial metabolomics data is seamlessly integrated with the workflow provided by the NEARDATA Metabolomics Data Space. In addition, we present the novel DATOMA App Store which is a cloud and edge computing based e-infrastructure for metabolomics data analysis, it offers a set of computational tools or applications covering the whole spectrum of metabolomics technologies based on mass spectrometry (MS), we will describe DATOMA in more detail later. Finally, the KIO Networks Data Center is included as a testbed platform fully adapted and capable of providing the necessary resources for the use of the different technologies presented.

The integration of the METASPACE platform with Lithops serverless analytics technology was

realized during the H2020 CloudButton project to offer scalable and elastic solutions from cloud architectures and is currently in production. At NEARDATA, we have focused on adapting the use of Lithops in METASPACE to replace the use of serverless functions with the Kubernetes backend. This modification is motivated by the introduction of confidential computing in the NEARDATA Metabolomics Data Space. The integration of SCONE with Lithops allows securing serverless data analysis resources for sensitive or confidential data processing.

Regarding the Control Plane components, the Data Broker will be in charge of orchestrating the TEEs deployed with SCONE to protect the executions with Lithops on METASPACE, additionally, the AI-based optimizations will help us to establish an efficient workflow on our data processing platform.

Next, we present DATOMA as the App Store of our NERADATA Metabolomics Data Space. DATOMA is a cloud computing web service solution that offers experimental scientists a fast and easy way to analyze mass spectrometry-based metabolomics data for free, while giving developers the opportunity to increase visibility and usability of their code.

DATOMA is intuitive, easy to use and free, so that researchers can focus on their research without the need to configure and orchestrate infrastructure or resources required for metabolomics data analysis. The cloud computing solution based on multi-cloud serverless programming will allow users to set-up and execute metabolomics data processing workflows over distributed/federated resources in the Cloud/Edge Continuum, facilitating easier collaboration between remote sites and introducing a standardised, reproducible, and accepted data analysis process.

DATOMA offers code developers the ability to share their code and tools making it more accessible to a wider audience of researchers, increasing the likelihood that the code will be used by others. DATOMA aims to be a comprehensive and accessible resource of data analysis tools helping to drive scientific progress.

Initially, DATOMA was conceived for users to upload to it their datasets to be processed through the metabolomics data analysis toolkit. The extreme sizes of currently existing metablomics data make it difficult for researchers to download these data from data sources for re-upload into our platform. To address this problem, DATOMA features integration with three metabolomics repositories widely used by the scientific community: METASPACE, Metabolights[17] and Metabolomics Workbench[18]. The three different repositories incorporate APIs for searching datasets from metadata. Likewise, the integration of these data sources with DATOMA is simple. DATOMA offers a data catalog from the metadata of the datasets found in each of the platforms, thus extending its range of metabolomic data to be processed. To avoid data duplication, datasets are only downloaded directly from the repositories within the job executions.

We present a last integration between DATOMA, METASPACE and Dataplug. DATOMA has been integrated with Dataplug to take advantage of dynamic partitioning to improve data throughput and reduce the data transferred between METASPACE and the jobs deployed in DATOMA. Specifically, a new library has been implemented in Dataplug for partitioning imzML data files for MS imaging processing.

We will validate the NEARDATA Metabolomics Data Space through the different KPIs defined at the beginning of the deliverable. The integration of Dataplug with DATOMA and METASPACE will be validated with *KPI-1 Significant performance improvements (data throughput, data transfer reduction) in the Extract-Transform-Load (ETL) phases validated with close data connectors on extreme data volumes*. As we have presented, Dataplug offers advantages for the extraction and consumption of data stored in object storage, leading to reduced data transfer and improved data throughput. Lithops has the ability to offer fully scalable and flexible serverless solutions depending on the resources required to process massive workloads. Therefore, it will be validated with the *KPI-3 Demonstrated resource auto-scaling for batch and stream data processing validated thanks to data-driven orchestration of massive workflows.* The incorporation of the confidentiality layer offered by SCONE on the Lithops serverless

[17] `https://www.ebi.ac.uk/metabolights`
[18] `https://www.metabolomicsworkbench.org/`

analysis platform will be validated from *KPI-4 High levels of data security and confidential computing validated using TEEs and federated learning in adversarial security experiments.* Finally, the repositories presented are widely used by the scientific community, with emphasis on METASPACE, the core component of our Metabolomics Data Space. At NEARDATA we aim to leverage this novel NEARDATA Metabolomics Data Space as a reference in the field of metabolomics. We will validate our NEARDATA Metabolomics Data Space through the *KPI-5 Demonstrated simplicity and productivity of the software platform validated with real user communities in International Health Data Spaces*. Finally, the EMBL metabolomics use case will allow us to validate the NEARADATA Metabolomics Data Space from a real metabolomics extreme data processing scenario.

6.2 Genomics Data Space

At NEARADATA we intend to contribute to a new International Health Data Spaces in the field of genomics to consolidate the new NEARDATA Genomics Data Space.

As the novel NEARDATA Genomics Data Space does not incorporate a core component, it is intended to ensure interoperability and integration of popular and widely used open source projects that we present in NEARADTA to create a reference architecture for extreme genomic data management, massive workload execution and confidentiality of sensitive data processing workflows.

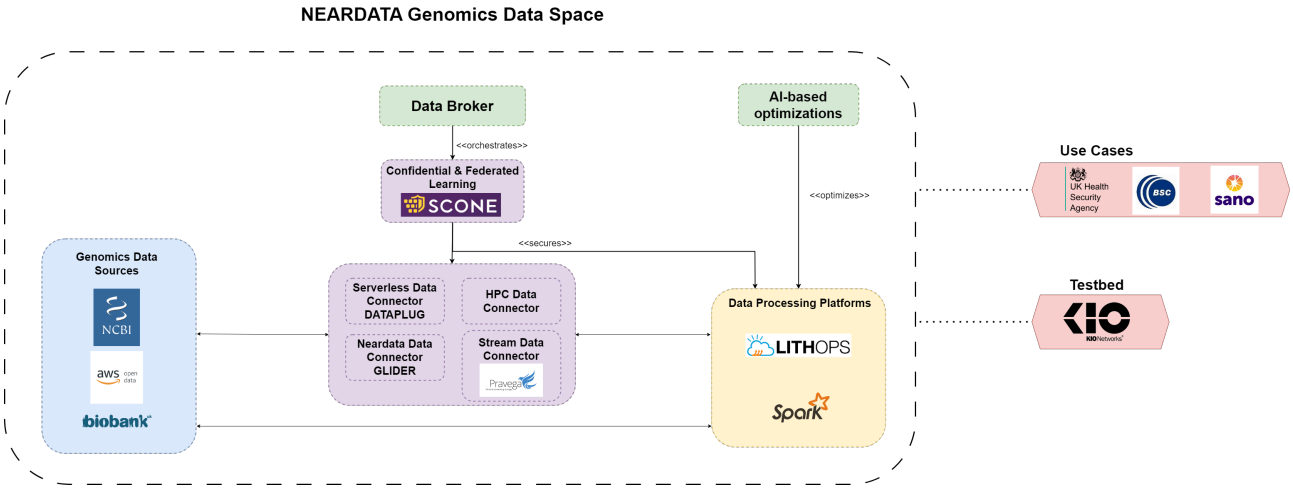

Figure 32: NEARDATA Genomics Data Space.

Figure 32 depicts the architecture of the NEARDATA Genomics Data Space. As all the International Health Data Spaces we present are based on our NEARDATA platform, we note that the architecture is similar to the previous NEARDATA Metabolomics Data Space. On the Control Plane module, we find the Data Broker component and the AI-based optimizations. As processing platforms we include Apache Spark together with Lithops. Additionally, we present open genomics data repositories such as NCBI, registry of Open Data on AWS[19] and UK Biobank. Within the Data Plane module is where we find some differences from the previous International Health Data Space. It should be noted that the Data Connectors we present in NEARDATA can be specialized in a single scientific field as is the case of the HPC Data Connector and the new Glider Neardata Data Connector that we will present later or belong to multiple fields such as Dataplug (Metabolomics and Genomics) and Pravega (Genomics and Surgomics). To guarantee the confidentiality of data and executions and to offer federated learning architectures we leverage the SCONE component. The KIO Networks Data Center provides the NEARDATA platform with the necessary resources to deploy and integrate all the components of the novel NEARDATA Genomics Data Space. Finally, we present up to three genomics use cases: Genomics use case (UKHS), Epistasis analytics in massive genomics datasets (BSC) and Transcriptomics Atlas use case/Federated Learning use case (SANO)

[19] `https://registry.opendata.aws/`

to validate with real use cases the NEARDATA Genomics Data Space.

In relation to the new integrations of technologies appearing in the NEARDATA Genomics Data Space, we can highlight the integration between Pravega and Lithops, where Lithops is used within the DELL Streaming Data Platform as a serverless function executor. Next, the integration between Pravega and SCONE is in our next steps in the development of the NEARDATA platform to secure and protect low-latency, real-time data processing and communication. Finally, we find the integration of two of the specialized Data Connectors in the use cases such as the HPC Data Connector introduced in the BSC use case that integrates with Apache Spark and the new Glider NEARDATA Data Connector that integrates with the UKHS use case.

The Glider NEARDATA Data Connector[20] is an ephemeral storage system incorporating near-data computation to reduce the volume of data transferred for our serverless architecture. Glider aims to improve communication between serverless computing stages, allowing data to "glide" smoothly through the processing pipeline rather than bouncing between different services. Glider achieves this by leveraging near-data-state execution of complex data-bound operations and an efficient I/O streaming interface.

Glider intercepts the data stored in its storage with novel storage actions to perform near-data computation before it is sent to the next stage of functions. For a more extended view of Glider we redirect the reader to deliverable *D5.1 First release of KPI benchmarks in all use cases and data connector libraries*.

This novel and specialized Data Connector is seamlessly integrated into the UKHS use case to store intermediate data while performing near-data computation to reduce the cost of data transfers in terms of data reduction and latencies.

The NEARDATA Genomics Data Space will be validated with the KPIs described for the NEARDATA platform. Data Connectors such as Dataplug and HPC Data Connectors will be validated through the *KPI-1 Significant performance improvements (data throughput, data transfer reduction) in the Extract-Transform-Load (ETL) phases validated with close data connectors on extreme data volumes*, both libraries offer capabilities for efficient genomic data preprocessing and transfer. Data Connectors that are described as systems such as Glider and Pravega are related to *KPI-1 Significant performance improvements (data throughput, data transfer reduction) in the Extract-Transform-Load (ETL) phases validated with close data connectors on extreme data volumes*, *KPI-2 - Significant data speed improvements (throughput, latency) in real-time video analytics validated using stream data connectors* and *KPI-3 Demonstrated resource auto-scaling for batch and stream data processing validated thanks to data-driven orchestration of massive workflows* for validation of data reduction and data throughput improvements along with real-time data speed improvements, and the scalability of resources offered by both distributed systems to cope with massive workloads. Data processing platforms such as Apache Spark and Lithops will be validated through *KPI-3 Demonstrated resource auto-scaling for batch and stream data processing validated thanks to data-driven orchestration of massive workflows*. Additionally, the TEEs deployed by SCONE to secure the executions and data management will be validated through *KPI-4 High levels of data security and confidential computing validated using TEEs and federated learning in adversarial security experiments.* Finally, the NEARDATA Genomics Data Space will be validated with *KPI-5 Demonstrated simplicity and productivity of the software platform validated with real user communities in International Health Data Spaces* from real users and use cases.

6.3 Surgomics Data Space

At NEARDATA we go beyond the International OMICs Data Spaces (Metabolomics and Genomics) to present the first Health Data Space on Surgery under the name NEARDATA Surgomics Data Space.

The NEARDATA Surgomics Data Space aims to become a reference as an open Health Data Space for Hospitals and research laboratories. Our NEARDATA partner NCT, expert in surgical data, presents the collaboration with Heidelberg Hospital together with two other associated hospi-

[20] `https://github.com/neardata-eu/glider-store`

tals. The interest shown by different hospitals has motivated the development of the NEARDATA Surgomics Data Space.

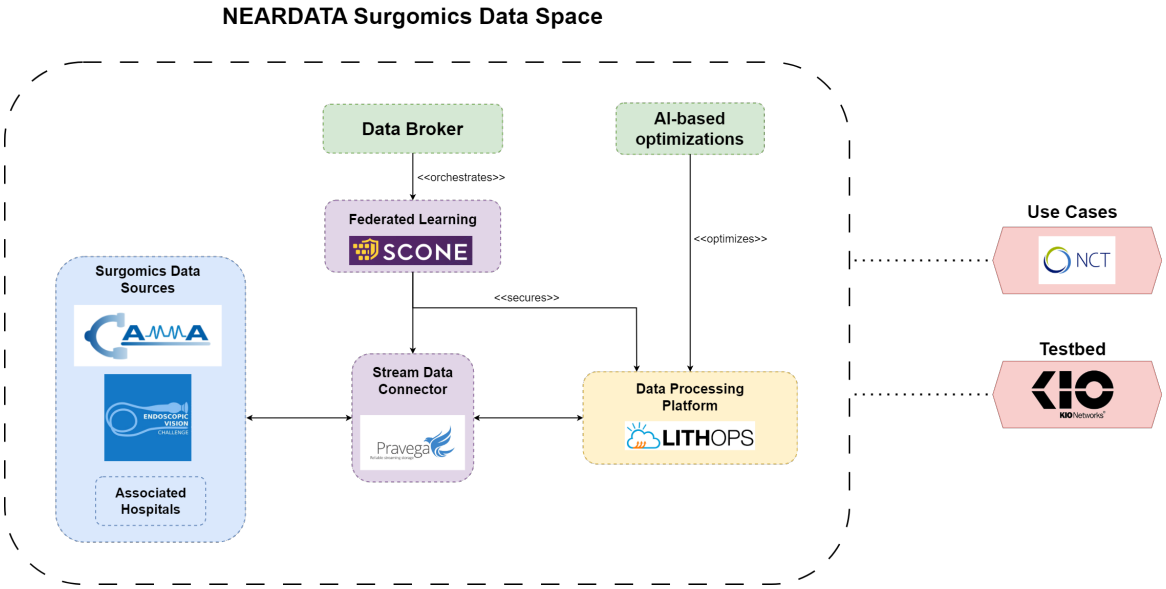

Figure 33: NEARDATA Surgomics Data Space.

Figure 33 shows the NEARDATA Surgomics Data Space. We can see that the core of our surgery data space is the main components of the novel XtremeHub platform that we present in NEARDATA, consisting of the SCONE (XtremeHub Security), Lithops (XtremeHub Compute) and Pravega (XtremeHub Streams) components. As in previous NEARDATA OMICs Data Spaces, we incorporate the Data Broker to orchestrate the confidential layers along with ia-based optimizations for the Lithops processing platform. We also introduce the Camma and Endoscopic Vision Challenge[21] data sources along with sensitive datasets offered by the different associated hospitals. In this way, the NEARDATA Surgomics Data Space allows establishing protocols and policies for the execution of Federated Learning architectures. Finally, we also include the KIO Networks Data Center to provide the necessary resources to deploy the different components and we present the surgery use case (NCT) to validate the data space with real users and use cases.

We can highlight the integrations of SCONE, Pravega and Lithops to initially form the NEARDATA XtremeHub and ensure low-latency, real-time data processing of sensitive surgical data.

Finally, we will validate the NEARDATA Surgomics Data Space from *KPI-1 Significant performance improvements (data throughput, data transfer reduction) in the Extract-Transform-Load (ETL) phases validated with close data connectors on extreme data volumes*, *KPI-2 - Significant data speed improvements (throughput, latency) in real-time video analytics validated using stream data connectors* and *KPI-3 Demonstrated resource auto-scaling for batch and stream data processing validated thanks to data-driven orchestration of massive workflows* on the XtremeHub Streams Pravega component. We will validate the XtremeHub Compute Lithops component through *KPI-3 Demonstrated resource auto-scaling for batch and stream data processing validated thanks to data-driven orchestration of massive workflows*. The XtremeHub Security SCONE component will be validated with *KPI-4 High levels of data security and confidential computing validated using TEEs and federated learning in adversarial security experiments.* Finally, the NEARDATA Surgomics Data Space will be validated with *KPI-5 Demonstrated simplicity and productivity of the software platform validated with real user communities in International Health Data Spaces* from the surgical use case presented by the NCT and real users from associated hospitals.

[21] `https://opencas.dkfz.de/endovis/challenges/2023/`

7 Conclusions

This deliverable presents the specifications of the NEARDATA architecture bringing the reader closer to understanding how the actual components that make up each module of the platform interact to create the desired workflows. In addition, new implementations of the components described theoretically in the previous deliverable are incorporated.

As the project is progressing as expected, we have been able to present integrations of the components already developed as early prototypes, where we show the interactions and combinations of the technologies to work correctly. In addition, we have presented the three NEARDATA International Health Data Spaces from the combination and adaptation of the different integrations.

In this way, we ensure that the NEARDATA platform is being implemented from the overall architecture proposed and that it meets the main objectives of the project.

We can highlight the following points as a conclusion:

- This deliverable allows to contextualize the NEARDATA project from the problems it wants to address through the proposed state-of-the-art research lines.
- The proposed architecture corresponds to the initial specifications described in the first deliverable. We combine the real components that we present in the NEARDATA platform to form four major modules. Building upon the theoretical specifications, we offer actual technical solutions.
- The components previously introduced in the initial NEARDATA architecture deliverable are briefly described to recall their use and operation in the platform. In addition, we have presented the novel Dataplug framework, which is based on the actual implementation of the Serverless Data Connector component which had benn described theoretically. In Dataplug we have developed different libraries for dynamic partitioning and data ingestion for different data processing platforms such as Ray, Dask and Lithops. Finally, we present the new HPC Data Connector, able to take advantage of high-performance computing platforms.
- We have observed the requirements and resources of our testbed in accordance with the architecture specifications.
- We have developed the first prototypes of the component integrations found in different modules of the NEARDATA platform. This section allows the reader to understand the real life cycle operation of the two life cycles presented in NEARDATA. We have described the combinations of components such as Lithops (Analytics) and Dataplug or Pravega (Data Connectors) that ensure the discovery, processing and analysis of extreme data. To provide secure data access and execution, we have added SCONE integration to form the novel XtremeHub of the NEARDATA platform.
- We have presented the three NEARDATA International Health Data Spaces in the fields of metabolomics, genomics and surgery from the combination of the different integrations of the NEARDATA platform components. In addition we have related the use cases to validate each of the International NEARDATA Data Spaces.

Overall, all these points guarantee the fulfillment of NEARDATA's main objective, to implement an extreme near-data analytics platform based on the novel XtremeHub platform. The platform can optimize data flows with high-performance near-data connectors and offers orchestration of secure workflows and confidential data access thanks to the Data Broker. As mentioned above, the architecture and the combination of components meet these requirements.

8 Appendix

8.1 NEARDATA APIs

8.1.1 Data Plane APIs

Dataplug. The role of Dataplug in NEARDATA is defined as a serverless Data Connector between the object storage and the various data processing platforms used in the use cases. Dataplug will allow an efficient access to unstructured data thanks to the following API:

API	API Call	Description
Cloud Object API	`from_path()`	Method used to create a Cloud Object from a path.
	`preprocess(backend, force)`	Method used to launch the preprocessing job for this Cloud Object on the specified preprocessing backend.
	`partition(strategy, args)`	Method to apply a partitioning strategy on this Cloud Object.
Data Slice API	`get()`	Method to evaluate the Data Slice, which will return the actual partition.

Table 1: Dataplug API.

HPC Data Connector. The role of this connector in NEARFDATA is defined as an HPC computational framework targeting the acceleration of use cases through the usage of HPC infrastructures, and in particular, the variant-interaction use case. This computation will be enabled through MPI-enabled API calls that allow for quick data loading, partitioning, learner and sorting among a large number of nodes. The API is defined as follows:

HPC Data Connector API	API Call	Description
HPC Loader	`read_labels()`	Method used to read patient's information.
	`convert_labels()`	Method used to prepare the data.
	`read_sample()`	Method used to read samples' information.
	`convert_samples()`	Method used to depurate samples' information.
HPC Partitioner	`get_mpicomm()`	Method to get MPI node information.
	`get_comb_files()`	Method to compute all the SNP-SNP combinations.
	`make_distribution()`	Method to deploy the data on the MPI processes.
HPC Learner	`filter_imputation()`	Method to catalogue values according to imputation quality.
	`transform_patients()`	Method to arrange every SNP-SNP pair as an interaction matrix.
	`applyMdrDict()`	Method to apply Multifactor Dimensionality Reduction MDR to every SNP-SNP combination.
	`get_risk_array()`	Method to count number of cases and number of controls for each column and estimate the high risk combinations.
	`get_error()`	Method to calculate the clasification error.
HPC Storer	`save_output()`	Method to save MDR-error to output directory.

Table 2: NEARDATA HPC Data Connector API.

8.2 NEARDATA Github repositories

Component Name	Module	Type	Repository
Lithops	Analytics	Component	https://github.com/neardata-eu/lithops-hpc-singularity-and-k8s-backend
Dataplug	Data Plane	Component & Integration with Lithops (Native)	https://github.com/neardata-eu/dataplug
METASPACE	Data Sources & Data Plane	Component	https://github.com/metaspace2020/metaspace
METASPACE & Lithops	Data Sources, Data Plane & Analytics	Integration	https://github.com/metaspace2020/Lithops-METASPACE
Pravega	Data Plane	Component	https://github.com/pravega/pravega
SCONE	Data Plane	Component	https://github.com/neardata-eu/scone-artifacts
Glider	Data Plane	Specific NEARDATA Genomics Data Space Component	https://github.com/neardata-eu/glider-store

Table 3: NEARDATA Github Repositories.

References

[1] J. You, J. Wu, X. Jin, and M. Chowdhury, "Ship compute or ship data? why not both?," in 18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21), pp. 633–651, USENIX Association, Apr. 2021.

[2] Gartner-Inc, "Data hubs, data lakes and data warehouses: How they are different and why they are better together." https://www.gartner.com/en/documents/3980938, 2020.

[3] Y. Moatti, E. Rom, R. Gracia-Tinedo, D. Naor, D. Chen, J. Sampe, M. Sanchez-Artigas, P. Garcıa-Lopez, F. Gluszak, E. Deschdt, F. Pace, D. Venzano, and P. Michiardi, "Too big to eat: Boosting analytics data ingestion from object stores with scoop," in 2017 IEEE 33rd International Conference on Data Engineering (ICDE), pp. 309–320, 2017.

[4] J. Sampé, M. Sánchez-Artigas, P. García-López, and G. París, "Data-driven serverless functions for object storage," in Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference, Middleware '17, (New York, NY, USA), p. 121–133, Association for Computing Machinery, 2017.

[5] J. Sampé, M. Sánchez-Artigas, G. Vernik, I. Yehekzel, and P. García-López, "Outsourcing data processing jobs with lithops," IEEE Transactions on Cloud Computing, vol. 11, no. 1, pp. 1026–1037, 2023.

[6] "Amazon s3 object lambda." https://aws.amazon.com/es/s3/features/object-lambda/.

[7] M. Kiran, P. Murphy, I. Monga, J. Dugan, and S. S. Baveja, "Lambda architecture for cost-effective batch and speed big data processing," in 2015 IEEE International Conference on Big Data (Big Data), pp. 2785–2792, 2015.

[8] K. Elmeleegy, "Piranha: optimizing short jobs in hadoop," Proc. VLDB Endow., vol. 6, p. 985–996, aug 2013.

[9] J. Ménétrey, C. Göttel, M. Pasin, P. Felber, and V. Schiavoni, "An exploratory study of attestation mechanisms for trusted execution environments," 2022.

[10] A. Szalay, "Extreme data-intensive scientific computing," Computing in Science & Engineering, vol. 13, pp. 34–41, nov 2011.

[11] A. K. Paul, W. Zhuang, L. Xu, M. Li, M. M. Rafique, and A. R. Butt, "Chopper: Optimizing data partitioning for in-memory data analytics frameworks," in 2016 IEEE International Conference on Cluster Computing (CLUSTER), pp. 110–119, 2016.

[12] Y. Q. X. S. L. Zhang and X. Li, "Workload-aware data placement for cloud computing," in North Carolina State University. Dept. of Computer Science, Tech. Rep, 2017.

[13] R. P. Abernathey, T. Augspurger, A. Banihirwe, C. C. Blackmon-Luca, T. J. Crone, C. L. Gentemann, J. J. Hamman, N. Henderson, C. Lepore, T. A. McCaie, N. H. Robinson, and R. P. Signell, "Cloud-native repositories for big scientific data," Computing in Science & Engineering, vol. 23, no. 2, pp. 26–35, 2021.

[14] Hobu-Inc, "Cloud optimized point cloud specification." https://copc.io/, 2022.

[15] "Cloud optimized geotiff." https://www.cogeo.org/, 2022.

[16] T. J. Skluzacek, R. Chard, R. Wong, Z. Li, Y. N. Babuji, L. Ward, B. Blaiszik, K. Chard, and I. Foster, "Serverless workflows for indexing large scientific data," in Proceedings of the 5th International Workshop on Serverless Computing, WOSC '19, (New York, NY, USA), p. 43–48, Association for Computing Machinery, 2019.

[17] R. Gracia-Tinedo, F. Junqueira, T. Kaitchuck, and S. Joshi, "Pravega: A tiered storage system for data streams," in Proceedings of the 24th International Middleware Conference, pp. 165–177, 2023.

[18] R. Gracia-Tinedo, F. Junqueira, B. Zhou, Y. Xiong, and L. Liu, "Practical storage-compute elasticity for stream data processing," in Proceedings of the 24th International Middleware Conference: Industrial Track, pp. 1–7, 2023.

[19] "Dell streaming data platform." `https://www.dell.com/en-us/dt/storage/streaming-data-platform.htm`, 2024.

[20] "Pravega - python client." `https://github.com/pravega/pravega-client-python`, 2024.

[21] "Pravega - rust client." `https://github.com/pravega/pravega-client-rust`, 2024.

[22] A. Palmer, P. Phapale, I. Chernyavsky, R. Lavigne, D. Fay, A. Tarasov, V. Kovalev, J. Fuchser, S. Nikolenko, C. Pineau, et al., "Fdr-controlled metabolite annotation for high-resolution imaging mass spectrometry," Nature methods, vol. 14, no. 1, pp. 57–60, 2017.

[23] S. Arnautov, B. Trach, F. Gregor, T. Knauth, A. Martin, C. Priebe, J. Lind, D. Muthukumaran, D. O'Keeffe, M. L. Stillwell, D. Goltzsche, D. Eyers, R. Kapitza, P. Pietzuch, and C. Fetzer, "SCONE: Secure linux containers with intel SGX," in 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), (Savannah, GA), pp. 689–703, USENIX Association, Nov. 2016.